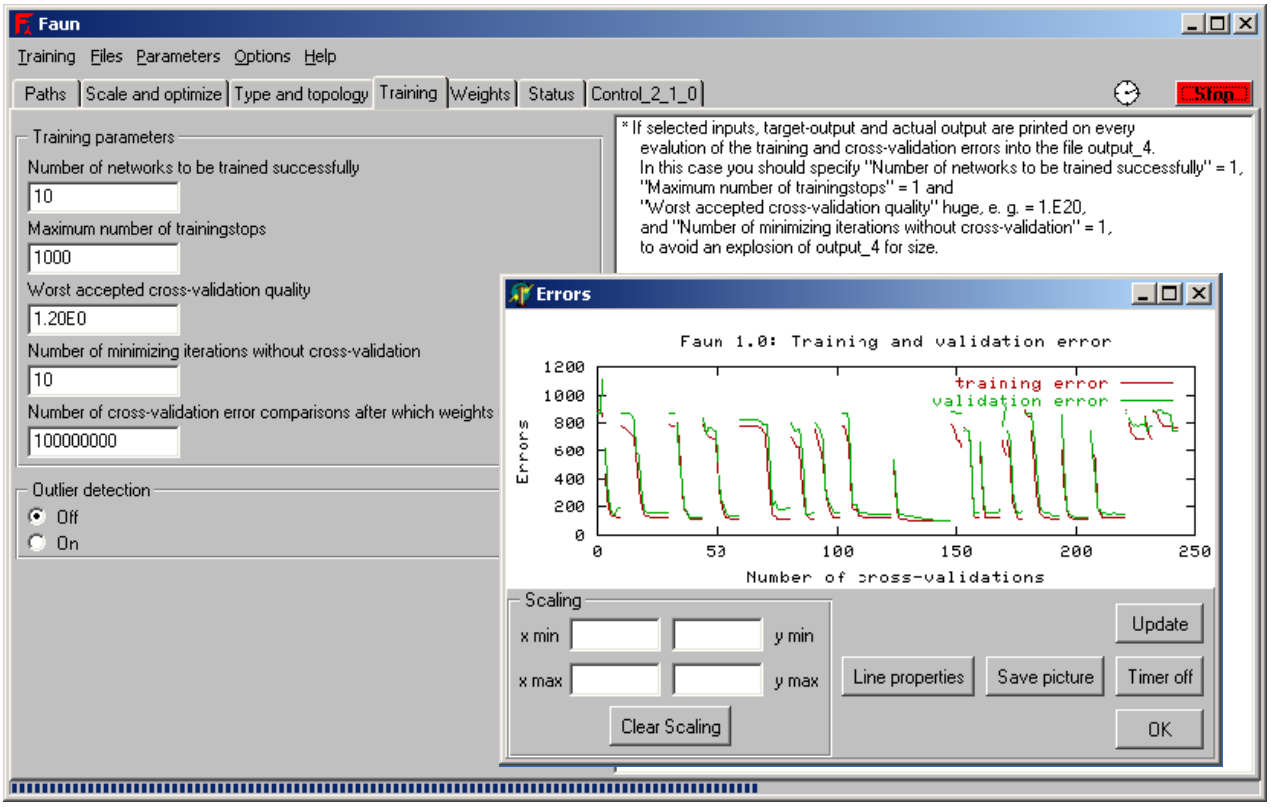
IWI Discussion Paper Series
16 (August 4, 2005)[1]

ISSN 1612-3646

FAUN 1.1 User Manual

Simon König[2], **Frank Köller**[3] and **Michael H. Breitner**[4]



[1] Copies or a PDF-file are available upon request: Institut für Wirtschaftsinformatik, Universität Hannover, Königsworther Platz 1, D-30167 Hannover, Germany (www.iwi.uni-hannover.de).

[2] Research Assistant and Lecturer (koenig@iwi.uni-hannover.de).

[3] Research Assistant and Lecturer (koeller@iwi.uni-hannover.de).

[4] Full Professor for Information Systems Research and Business Administration (breitner@iwi.uni-hannover.de).

Contents

1 Introduction

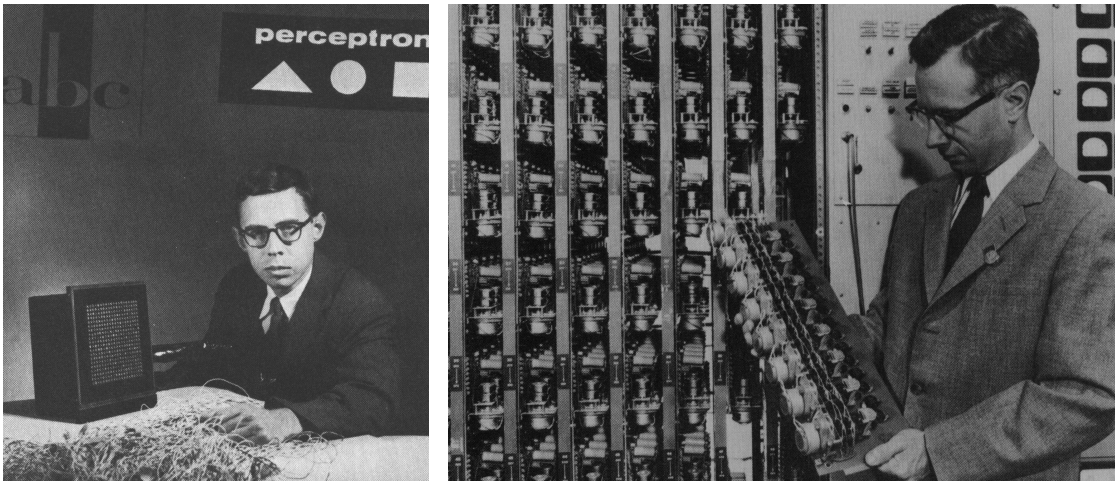


Figure 1.1: Frank Rosenblatt with his Mark I Perceptron in 1958

Neurocomputation starts in 1957/58 when Frank Rosenblatt, Charles Wightman and their co-workers build the Mark I Perceptron at the Massachusetts Institute of Technology (MIT). The neurocomputer Mark I Perceptron is the very first chapter of a dream-come-true story. Intelligence in machines – called artificial intelligence (AI) – is born. For the first time researchers try to clone intelligence of men: Results are quite poor and futurology says that it will be 2025 or 2030 before computers will reach human intelligence. The Mark I Perceptron electromechanically works with 512 motor driven potentiometers. The potentiometers represent the weights in synapses between neurons in an artificial, extremely tiny brain called artificial neural network (ANN). The resistances in the potentiometers can be adapted to train the ANN. Training or learning rules and methods used in the late 1950s are very poor. Nevertheless, finally Mark I Perceptron recognizes 10 simple figures in a 20 by 20 pixel array.

Today's neurocomputation usually is based on complete software emulation and is therefore often called neurosimulation. Inputs, outputs, neurons, synapses and weights are implemented in software. The neurosimulator FAUN (Fast Approximation with Universal Neural networks) enables supervised learning with 3- and 4-layered perceptrons and also radial basis functions. A FAUN user has to provide patterns, i. e. input-output pairs explaining a mathematical relation. Then ANNs are trained to learn the relation with a black-box approach. A well trained ANN is a mathematical function which approximates the output of the patterns sufficiently accurate. Furthermore this ANN

reasonably interpolates and extrapolates between the patterns (generalization). Here the word "reasonable" summarizes different quality factors for the trained ANN, i. e. the mathematical function, see [4] and the Appendix for details and examples. E. g. highly frequent oscillations which do not correspond to the patterns should be avoided.

Shortly summarized FAUN neurosimulator highlights are:

1. The software development is based on today's software quality principles:
   a) Suitable, correct and adequate functionality;
   b) High reliability, i. e. stability, error tolerance and restart ability;
   c) User friendly documentation, self-learning and ergonomics;
   d) Good overall performance and moderate allocation of resources, e. g. Cache or RAM;
   e) Good maintainability, i. e. readable, changeable and testable source code;
   f) High portability, i. e. easy installation, high compatibility with different hardware and operating systems and high convertibility.
2. The ANN types and topologies supported are among the most powerful and worldwide accepted for real life problems. The training and learning algorithms are based on approved numerical methods for constrained optimization problems and nonlinear least-squares problems, i. e. sequential quadratic programming (SQP) methods and generalized Gauß-Newton (GGN) methods. An ANN's training often is very difficult and computationally very expensive. Thus the training and learning time for ANNs are significantly shortened compared to other neurosimulators.
3. Various advanced graphics enables an online or posteriori supervision and analysis of an ANN training and learning. Special graphics helps, e. g., to detect outliers in the patterns.
4. The source code generator produces ready-to-use code for the trained ANNs in different computer languages. MAPLE output can be used to calculate or visualize in a powerful computer algebra system.
5. FAUN runs locally on most Windows, UNIX and LINUX computers. With the help of the WWW-frontend (thin client implementation for a WWW-browser) an ANN training on a LINUX compute server can be controlled remotely and asynchronously.
6. A FAUN-HPC (high performance computing) family for parallel, vector and grid computers is under development since 1999. FAUN-HPC prototypes use PVM, MPI or individual networking protocols for online and batch training and learning of ANNs. Challenging problems needing CPU-month or even CPU-years computing time already have been solved.

Questions, problems and proposals should be emailed to the FAUN research, development and support team: Michael H. Breitner (`breitner@iwi.uni-hannover.de`), Frank Köller (`koeller@iwi.uni-hannover.de`), Simon König (`koenig@iwi.uni-hannover.de`) and Hans-Jörg von Mettenheim (`mettenheim@iwi.uni-hannover.de`).

2 FAUN User Manual

2.1 FAUN usage with GUI

2.1.1 Path settings

As shown in figure 2.1 several paths have to be given. The Faun program is assumed to be in the Faun subdirectory of the installation directory; all relevant data are written into the data directory (if it does not exist, it will be created):

`control_and_output_files` This subdirectory contains the control files that determine Faun's behaviour. They will be generated by the GUI when you start Faun. For a description of possible parameters see section 2.1.2.

Output files contain data such as training and validation errors generated by Faun.

In addition, Faun generates several messages, e. g. about parameter settings and number of patterns for training and validation. These messages are stored in the file `faun.log`.

`data_files` This subdirectory holds training and validation data.

As the data directory contains all necessary information that belongs to a Faun run, you can backup the whole directory to be able to reproduce the results (see section 2.2.2).

Note that when changing the path to the Faun program the paths to the output files are adjusted accordingly. The output files are used to generate graphics that allow you to analyze the resulting networks. So if you change these paths to analyze networks from different Faun runs do not forget to adjust these settings when you want to analyze the output of a new Faun run.

The path settings for controlfiles are only used when you want to load or save controlfiles (see section 2.2.2). When starting Faun, the parameter values are always taken out of the GUI and not out of a specified controlfile.

When you have specified paths for training and validation data, these are copied into the Faun directory before Faun is started (see section 2.2.1). Otherwise it is assumed that these files are already in the Faun program directory so no error will be reported.

2.1.2 Parameter settings

All parameter settings are grouped on these tab sheets:

1. Scale and optimize
2. Type and topology

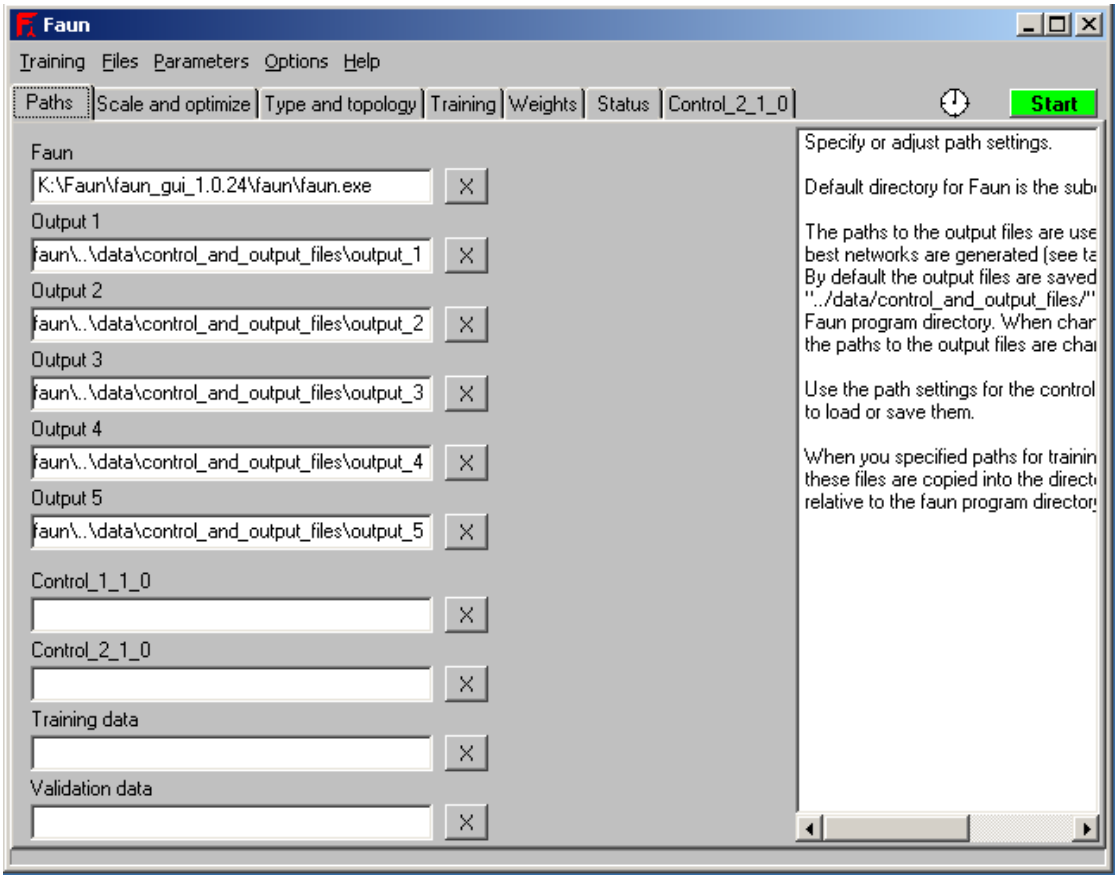


Figure 2.1: Main window with buttons partly to open FAUN subwindows: "Training" to start/stop FAUN, "Files" to load examples/store results, "Parameters" to check (all) parameters, "Options" to choose interface preferences, "Help" for the manual, "Paths" to set the paths for the input/output files, "Scale and optimize" to choose the patterns' scaling/the training method, "Type and topology" to set the network's number of neurons/topology, "Training" to set the training parameters, "Weights" to set the weights' initialization and boxes, "Status" to monitor a FAUN run graphically and via log-files, and "Control_2_1_0" for expert users to change parameters of the chosen training/optimization method. See also figure 2.2.

3. Training
4. Weights
5. Control_2_1_0

The last tab sheet shows options for the optimizer and should not be altered unless you really know what you are doing. See figure 2.2 for how the training parameters are displayed.

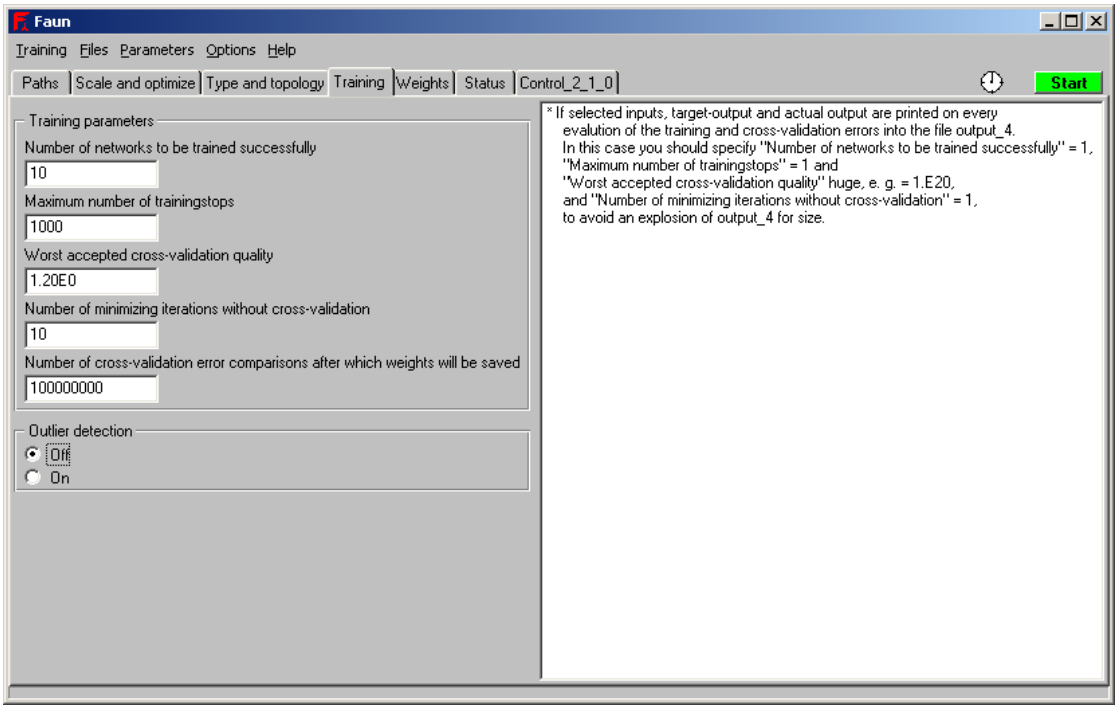


Figure 2.2: Main window showing training parameters. See also figure 2.1.

On the right of every tab sheet a description is displayed as soon as the according input field for a parameter is activated. This control is resizeable but does not overlay other controls. You can increase the size of the main window to gain more space for resizing the control showing the help texts.

In the following all parameters are described in detail.

Scale and optimize

Pattern scaling (1a) Select scaling:

Parameter setting	Value
On	1
Off	0

The first lines in `control_1_1_0` have the form

```
<parameter 1a> <parameter 1b> <parameter 1c>
<scalingmethod> <target-interval-minimum> <target-interval-maximum>
<scalingmethod> <target-interval-minimum> <target-interval-maximum>
...
```

`<scalingmethod>` can be linear or logarithmic:

Parameter setting	Value
linear	0
logarithmic	1

If the scaling button is not checked then FAUN uses the files `training_scaled` and `validation_scaled`, otherwise FAUN uses the files `training_unscaled` and `validation_unscaled` and generates the scaled pattern files `training_scaled` and `validation_scaled`.

For numerical stability all components x_i of the input vector $\boldsymbol{x} \in \mathbb{R}^{n_e}$ must be transformed to the same interval by scaling and equilibration. Then all components x_i are equal valued in the error functions. A recommended interval for a scaling of the input vector is $[-1, 1]^{n_e}$; a recommended interval for a scaling of the output vector is smaller than $[-1, 1]^{n_a}$, e. g. $[-0.95, 0.95]^{n_a}$, dependent of the approximation problem.

For the scaling the linear transformation

$$x_{i,\text{scaled}} := l\frac{x_i - x_{i,\min}}{x_{i,\max} - x_{i,\min}} - \frac{l}{2}, \quad i = 1, 2, \ldots n_e \tag{2.1}$$

or logarithmic transformation

$$x_{i,\text{scaled}} := l\frac{\ln(x_i) - \ln(x_{i,\max})}{\ln(x_{i,\max}) - \ln(x_{i,\min})} + \frac{l}{2}, \quad i = 1, 2, \ldots n_e \tag{2.2}$$

is used independently for all inputs, where l is the length of the interval and n_e is the number of input neurons or input variables, respectively. Scaling of output variables works analogously, but with a smaller interval. Logarithmic scaling is useful if *relative* changes of the components should be transformed to *absolute* changes.

With $n_e = 6$, linear scaled on $[-1, 1]$, and $n_a = 1$ (one output neuron), linear scaled on $[-0.95, 0.95]$ the following lines have to be set in the control file:

```
1 0 0
0 -1.E0 1.E0
0 -1.E0 1.E0
0 -1.E0 1.E0
0 -1.E0 1.E0
0 -1.E0 1.E0
0 -1.E0 1.E0
0 -0.95E0 0.95E0
```

In doubt use linear scaling with intervals $[-1, 1]^{n_e}$ and $[-0.95, 0.95]$, respectively, and only one output, i. e. $n_a = 1$.

Optimization method (1b) Approved numerical methods for constrained nonlinear least-squares problems arising here are (adapted) sequential quadratic programming

(SQP) methods and generalized Gauß-Newton (GGN) methods which can exploit special structures. SQP and GGN methods automatically can overcome most of the training problems of neural networks, e. g. flat spots or steep canyons of the training datas' error function. At the time the SQP methods NPSOL and NLSSOL from P. E. Gill, UC San Diego, are implemented, see [6, 8, 9].

Parameter setting	Value
NPSOL	0
NLSSOL	1
DFNLP (not yet implemented)	2
NLSCON (not yet implemented)	3

NPSOL should be preferred usually.

Type and topology

Neural network type (1c) Typical ANNs like 3-layered perceptrons (1 hidden layer), 4-layered perceptrons (2 hidden layers) and radial basis functions ANNs (RBF-ANNs) are implemented, see [5, 7, 10, 11, 12, 13, 14, 15].

Parameter setting	Value
3-layered perceptron	0
4-layered perceptron	1
radial basis functions network	2

Exemplarily 3-layered perceptrons, see figure 2.3, have an auxiliary neuron N_0 (bias neuron) with $x_{k,0} := \frac{1}{2}$ where $k = 1, 2, \ldots, n_p$ and n_p denotes the number of training and validation patterns, with input neurons $N_1, \ldots, N_{n_e}$, with hidden neurons $N_{n_e+1}, \ldots, N_{n_e+n_2+1}$ (N_{n_e+1} is the auxiliary bias neuron in the hidden layer), where n_2 is the number of hidden neurons and $n_2 \geq 1$ (usually $n_2 \leq 10$ is advisable), with output neurons $N_{n_e+n_2+2}, \ldots, N_{n_e+n_2+n_a+1}$[1] and with the hyperbolic tangent transfer function tanh are defined by

$$
\begin{aligned}
a_{k,i} &:= x_{k,i}, && \text{for } i = 0, 1, \ldots, n_e, \\
a_{k,i} &:= \tanh\left(\sum_{j=0}^{n_e} w_{j,i} a_{k,j}\right), && \text{for } i = n_e + 1, \ldots, n_e + n_2 + 1, \\
a_{k,i} &:= \tanh\left(\sum_{j=n_e+1}^{n_e+n_2+1} w_{j,i} a_{k,j}\right), && \text{for } i = n_e + n_2 + 2, \ldots, n_e + n_2 + n_a + 1, \\
& && \text{with } k \in I_t \cup I_v.
\end{aligned}
$$

[1] Generally only one output neuron is advisable ($n_a = 1$), i. e. separate neural networks for each output should be trained to facilitate the training significantly.

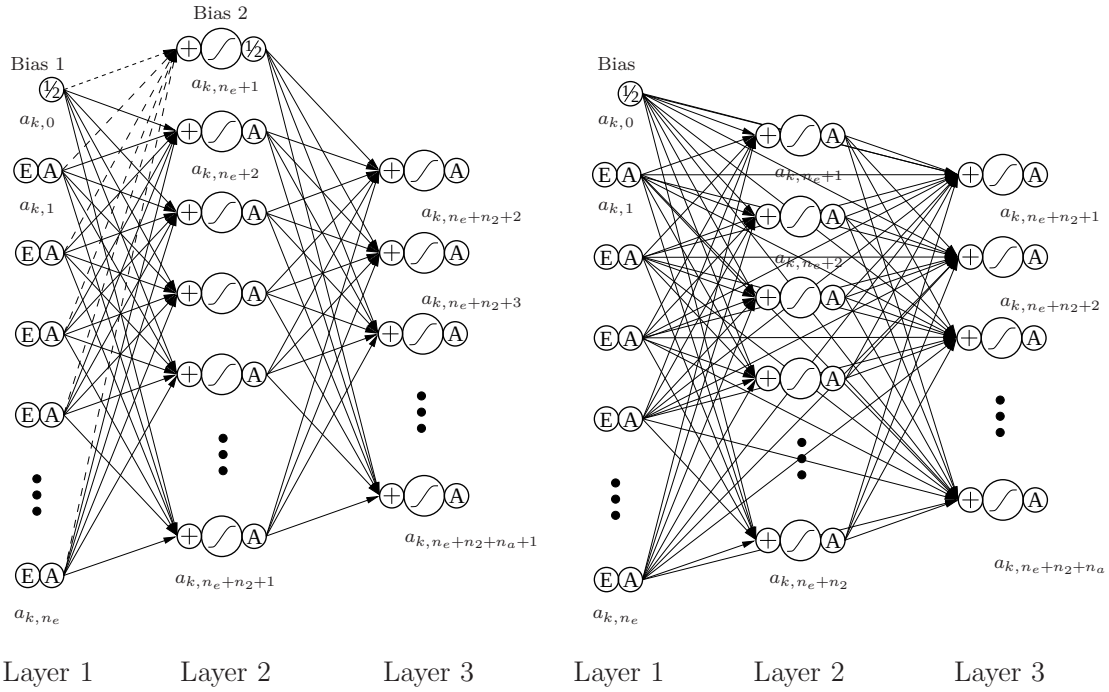


Figure 2.3: 3-layered perceptrons (input layer, 1 hidden layer with n_2 neurons, output layer) without shortcuts (left) and with shortcuts (right).

N_i's output for pattern $\boldsymbol{x}_k, \boldsymbol{y}_k$) is denoted by $a_{k,i}$.

With shortcut connections enabled 3-layered perceptrons are defined by

$$
\begin{aligned}
a_{k,i} &:= x_{k,i}, && \text{for } i = 0, 1, \ldots, n_e, \\
a_{k,i} &:= \tanh\left(\sum_{j=0}^{n_e} w_{j,i} a_{k,j}\right), && \text{for } i = n_e + 1, \ldots, n_e + n_2, \\
a_{k,i} &:= \tanh\left(\sum_{j=0}^{n_e+n_2} w_{j,i} a_{k,j}\right), && \text{for } i = n_e + n_2 + 1, \ldots, n_e + n_2 + n_a, \\
& && \text{with } k \in I_t \cup I_v,
\end{aligned}
$$

note the missing bias neuron in the hidden layer: it can be omitted, because with shortcut connections enabled the bias neuron in the input layer influences the output neurons.

4-layered perceptrons have auxiliary neurons in the input layer (N_0) and in the first and second hidden layer (N_{n_e+1} and $N_{n_e+n_2+2}$). These perceptrons are defined by

$$
\begin{aligned}
a_{k,i} &:= x_{k,i}, && \text{for } i = 0, 1, \ldots, n_e, \\
a_{k,i} &:= \tanh\left(\sum_{j=0}^{n_e} w_{j,i} a_{k,j}\right), && \text{for } i = n_e + 1, \ldots, n_e + n_2 + 1, \\
a_{k_i} &:= \tanh\left(\sum_{j=n_e+1}^{n_e+n_2+1} w_{j,i} a_{k,j}\right), && \text{for } i = n_e + n_2 + 2, \ldots, n_e + n_2 + n_3 + 2 \\
a_{k,i} &:= \tanh\left(\sum_{j=n_e+n_2+2}^{n_e+n_2+n_3+2} w_{j,i} a_{k,j}\right), && \text{for } i = n_e + n_2 + n_3 + 3, \ldots, \\
& && \qquad n_e + n_2 + n_3 + n_a + 2, \\
& && \text{with } k \in I_t \cup I_v,
\end{aligned}
$$

if shortcut connections are disabled, and with shorcuts enabled, respectively,

$$
\begin{aligned}
a_{k,i} &:= x_{k,i}, && \text{for } i = 0, 1, \ldots, n_e, \\
a_{k,i} &:= \tanh\left(\sum_{j=0}^{n_e} w_{j,i} a_{k,j}\right), && \text{for } i = n_e + 1, \ldots, n_e + n_2 + 1, \\
a_{k_i} &:= \tanh\left(\sum_{j=n_e+1}^{n_e+n_2+1} w_{j,i} a_{k,j}\right), && \text{for } i = n_e + n_2 + 2, \ldots, \\
& && \qquad n_e + n_2 + n_3 + 1 \\
a_{k,i} &:= \tanh\left(\sum_{j=0}^{n_e} w_{j,i} a_{k,j} + \sum_{j=n_e+n_2+2}^{n_e+n_2+n_3+1} w_{j,i} a_{k,j}\right), && \text{for } i = n_e + n_2 + n_3 + 2, \ldots, \\
& && \qquad n_e + n_2 + n_3 + n_a + 1, \\
& && \text{with } k \in I_t \cup I_v.
\end{aligned}
$$

When shortcut connections are enabled the bias neuron in layer 3 can be omitted.

For a detailed description of radial basis function networks see updated Venia Legendi thesis of Michael H. Breitner (see also item “Publications” on `http://www.iwi.uni-hannover.de`).

All weights $w_{j,i}$ of the weight matrices

$$W_{12} = \begin{pmatrix} w_{0,n_e+1} & \cdots & w_{n_e,n_e+1} \\ \vdots & \ddots & \vdots \\ w_{0,n_e+n_2+1} & \cdots & w_{n_e,n_e+n_2+1} \end{pmatrix}, \tag{2.3}$$

$$W_{23} = \begin{pmatrix} w_{n_e+1,n_e+n_2+2} & \cdots & w_{n_e+n_2+1,n_e+n_2+2} \\ \vdots & \ddots & \vdots \\ w_{n_e+1,n_e+n_2+n_a+1} & \cdots & w_{n_e+n_2+1,n_e+n_2+n_a+1} \end{pmatrix}, \tag{2.4}$$

with $W_{12} \in \mathbb{R}^{n_2+1,n_e+1}$ (layer 1 → layer 2) and $W_{23} \in \mathbb{R}^{n_a,n_2+1}$ (layer 2 → layer 3), are trainable except w_{0,n_e+1}. In case of shortcuts $W_{13} \in \mathbb{R}^{n_a,n_e+1}$ exists analogously and the bias neuron in the hidden layer is omitted:

$$W_{13} = \begin{pmatrix} w_{0,n_e+n_2+1} & \cdots & w_{n_e,n_e+n_2+1} \\ \vdots & \ddots & \vdots \\ w_{0,n_e+n_2+n_a} & \cdots & w_{n_e,n_e+n_2+n_a} \end{pmatrix}$$

For enabling shortcuts see parameter 8 ("Shortcut connections")).

4-layered perceptrons have an additional hidden layer and therefore is $W_{23} \in \mathbb{R}^{n_3+1,n_2+1}$ (layer 2 → layer 3) and an extra weight matrix $W_{34} \in \mathbb{R}^{n_a,n_3+1}$ (layer 3 → layer 4), where n_2 denotes the number of neurons in the first hidden layer and n_3 the number of neurons in the second hidden layer. If shortcuts are enabled then exists $W_{14} \in \mathbb{R}^{n_a,n_e+1}$. Usually only one output neuron is used ($n_a = 1$).

In doubt take 3-layered perceptrons with only few hidden neurons and only one output neuron.

For a detailed description of radial basis function networks see updated venia legendi sisis of Michael H. Breitner (see also item "Publications" on `http://www.iwi.uni-hannover.de/`).

Shortcut connections (8) Direct connections from input neurons to output neurons are called shortcuts or shortcut connections. Shortcuts often are useful in case of a dominant linear part in the input/output relation.

Parameter setting	Value
On	1
Off	0

In doubt switch shortcuts off.

Number of neurons in the input layer (10) This parameter n_e must be adapted to the problem. The bias neuron does not count.

Number of neurons in the hidden layers (11a and 11b) The number of neurons n_2 in hidden layer 2 and n_3 in layer 3 (only 4-layered perceptrons) without counting the bias neuron(s).

Start problem solving with only few internal neurons, i. e. $1, 2$ or 3, and generally obey $n_2 > n_3$.

Number of neurons in the output layer (12) Defines the number n_a of components of the output vector.

An approximation problem with $n_a > 1$ should be divided in n_a approximation problems with only one-dimensional output to facilitate the training with FAUN significantly.

Number of parallely trained networks (3) Define the number of parallely trained networks, i. e. the possibility to parallely train several networks with the same patterns and the same topology (> 1 should not be used).

Training

Number of networks to be trained successfully (2) For a so called successfully trained neuronal network holds

$$\frac{\varepsilon_v/n_v}{\varepsilon_t/n_t} < (\text{worst, acceptable validation quality})^2, \tag{2.5}$$

where the worst, acceptable validation quality is defined by parameter 5 ("Worst accepted cross-validation quality"). The training and validation error functions are given by (here w. l. o. g. for simplicity given for $n_a = 1$ and a 3-layered perceptron without shortcuts)

$$\varepsilon_t(W) := \frac{1}{2} \sum_{k \in I_t} \left(a_{k,n_e+n_2+2}(W) - y_k\right)^2 \text{ and} \tag{2.6}$$

$$\varepsilon_v(W) := \frac{1}{2} \sum_{k \in I_v} \left(a_{k,n_e+n_2+2}(W) - y_k\right)^2. \tag{2.7}$$

For3-layered perceptrons with shortcuts, for 4-layered perceptrons and for radial basis functions networks analogous definitions of ε_t and ε_v hold. For $n_a > 1$ a sum over all output neurons must be taken.

Recommended values are 100 for easy problems up to 10000 or even 100000 for (extremely) difficult problems. The more neural networks with different weight initializations are trained, the more likely it is to train a good network, i. e. to find a neighborhood of a good local minimum of ε_t. In doubt first compute only 100 successfully trained networks.

Maximum number of trainingstopps (4) Cross-validation will be regularly executed during the iterative minimization of the training error ε_t, see parameter 6 ("Number of minimizing iterations without cross-validation"), and the cross-validation error will be compared with the cross-validation errors of the last two stops. If the cross-validation error increases two times, the iterative training of the network will be stopped. Note that thus the training usually stops before a local minimum of ε_t is reached (prevention of overtraining). It will be tested then if the network is trained successfully, see parameter 2 ("Number of networks to be trained successfully") and parameter 5 ("Worst accepted cross-validation quality"). If necessary a new network will be initialized after a complete training of a network.

For a test of the weight initializations and of the initial error set e. g. parameter 4 = 0 and parameter 2 = 100000. Recommended values are 1000 for easy problems up to 10000 for (extremely) difficult problems. In doubt choose a maximum of 1000 minimizing iterations without cross-validation.

Worst accepted cross-validation quality (5) The worst, acceptable cross-validation quality is used to reject overtrained, i. e. not successfully trained, neural networks, see parameter 2 ("Number of networks to be trained successfully"). To train into the minimum of the training error function it is necessary that this parameter is large, e. g. 1.E20.

Weight upgrades $W_{\text{new}} - W_{\text{old}}$ can be calculated with any minimization algorithm, e. g. with usually favored first derivative methods such as the steepest descent or with second derivative methods such as the Newton method. For first derivative methods we have the iterative sequence

$$W_{\text{new}} = W_{\text{old}} + \eta \left(\varepsilon_t \left(W_{\text{old}}\right), \text{grad}_W \, \varepsilon_t \left(W_{\text{old}}\right)\right) \Delta W \left(\varepsilon_t \left(W_{\text{old}}\right), \text{grad}_W \, \varepsilon_t \left(W_{\text{old}}\right)\right) \quad (2.8)$$

with search direction ΔW and with step length η. Approved numerical methods for constrained nonlinear least-squares problems, see Eq. (2.6), are sequential quadratic programming (SQP) methods and generalized Gauß-Newton (GGN) methods which can exploit the special structure of the Hessian matrix of ε_t. SQP and GGN methods automatically can overcome most of the training problems of perceptrons, e. g. flat spots or steep canyons of the error function ε_t. But SQP and GGN methods only converge towards one of the usually very many local minima of ε_t. SQP and GGN methods usually approximate the Hessian matrix of ε_t by finite-differences and update formulas, i. e. become first derivative methods, and can deal with box constraints, linear constraints and smooth nonlinear constraints. Advantages of these methods are:

- A much better search direction ΔW is calculated in comparison to common training methods, e. g. $\Delta W := \text{grad}_W \, \varepsilon_t$ for the gradient method (backpropagation);

- The step length η is optimized permanently in contrast to common training methods with fixed step length. Thus the number of learning steps is reduced significantly (factor 10, 100, 1000, ...);

- Only ε_t, $\text{grad}_W\, \varepsilon_t$ and ε_v are required which mainly can be computed by very fast matrix operations. For other neural network topologies, e. g. radial basis functions, an efficient code for $\text{grad}_W\, \varepsilon_t$ also can be deduced by automatic differentiation;
- Maximum and minimum of each weight can be set easily (box constraints, see parameter 13a ("Topology setting"));
- The total curvature of the neural network can be constrained (prevention from neural network oscillations);
- Convexity and monotonicity constraints can be set.

The worst accepted cross-validation quality should be selected in a way that 80 % to 95 % of the trained networks after the last stop are judged as not successfully trained and are initialized again. Recommended values are between 0.5 and 1.5, dependent on the problem, especially dependent on the training set P_t and the cross-validation set P_v.

Number of minimizing iterations without cross-validation (6) Recommended values are 1 for very easy problems to 10 for difficult problems and up to 100 for very difficult problems. The quality of successfully trained networks depends heavily upon this parameter and also on parameter 5 ("Worst accepted cross-validation quality"). In doubt choose 5 or 10.

Training error output, the validation error output and the information parameter of the optimizer are saved in the file `output_1`.

Number of cross-validation error comparisons after which weights will be saved (7) After a network is successfully trained the weights are printed into file `output_2`. If the weights for the current network should be printed only at the end of a successfully training this parameter must be large, e. g. 1000000000.

In case the weights should be printed for any training stop this parameter should be 1.

The output of the weights with errors and the information parameter of the optimizer are always printed into file `output_2` which may become very large.

Outlier detection (9) The approximation quality of an optimal approximation function depends heavily on outliers and unusual cluster distributions. Thus often it is necessary to find and eliminate them.

FAUN should be initialized with some neural networks for testing purposes and the overall error should be trained into a local minimum using combined training and validation patterns. Usually few neurons, often only one, two or three, are sufficient. For the neural network with the smallest overall error inputs, reference output(s) and actual output(s) should be printed. This is done on every evaluation of the training and cross-validation errors into the file `output_4` if switched on.

For detecting outliers with Gnuplot reference − actual output has to be computed and depicted, see also section 2.1.3. In this case you should specify parameter $2 = 1$,

parameter 4 = 1 and parameter 5 large, e. g. 1000000000, and parameter 6 = 1, to avoid an explosion of `output_4`'s size.

Parameter setting	Value
On	1
Off	0

Weights

Topology setting (13a) and Random seed (13b and 13c) All parameters have to be entered in one line, e. g.

```
<topology setting> <random seed 1> <random seed 2>
```

where `random seed 1` and 2 should be prime numbers between 11 and 997.

Parameter setting	Value
Explicit setting of topology and weights (first network only, others randomly initialized)	0
Explicit setting of topology and random initialization of weights	1
Automatic generation of a fully connected network and random initialization of weights	2

Explicit setting of topology and weights All weights must be specified in any sequence where every line looks like

```
<weight> <minimum> <maximum> <initialization-minimum> \
               <initialization-maximum> <weight matrix> <source> <target>
```

e. g.

```
0.91836352E+01 -0.1E+03 0.1E+03 -0.1E+02 0.1E+02 1 1 5
```

For the second and all further networks the random initialization generally uses the numbers from [initialization-mininimum,initialization-maximum] with random seed 1 and 2. This enables, e. g., also to create incompletely connected networks and specialized topologies.

Explicit setting of the topology and random initialization of weights All weights must be specified in any sequence (see above). For every network the random initialization uses the numbers from [initialization-mininimum,initialization-maximum] with random seed 1 and 2. This enables, e. g., also to create incompletely connected networks and specialized topologies.

Automatic generation of a fully connected network and random initialization of all weights Only one line (perceptrons) has to be given, e. g.

```
0.1E+01 -0.1E+03 0.1E+03 -0.1E+02 0.1E+02 1 1 5
```

or only three lines (RBF-networks), e. g.

```
0.1E+01  0.1E-10 0.1E-02  0.1E-10 0.1E-02 1 1 6 (diameter)
0.1E+01 -0.1E+01 0.1E+01 -0.1E+01 0.1E+01 1 2 6 (centercomp.)
0.1E+01 -0.1E+03 0.1E+03 -0.1E+02 0.1E+02 1 1 5 (weigths)
```

The diameter of the radial basis functions networks must be positive. For every network and all weights, respectively, the random initialization uses the numbers from [initialization-mininimum,initialization-maximum] with random seed 1 and 2.

Control_2_1_0

Optimizer's print channel Default recommendation: Do not alter parameter unless you are an expert user.

Optimizer's verify level Default recommendation: Do not alter parameter unless you are an expert user.

Optimizer's derivative level Default recommendation: Do not alter parameter unless you are an expert user.

Optimizer's major print level Default recommendation: Do not alter parameter unless you are an expert user.

2.1.3 Status

Overview

This tab sheet shows the log file of the last Faun run. Here you can verify your settings like topology, number of neurons per layer, number of training and validation patterns. When Faun finishes the log file contains also the time Faun needed to compute the networks.

Top ten networks

By pressing the button labeled "Generate" the file "`output_2`" (as specified on the "Paths" tab sheet) will be analyzed to determine the ten best networks. These will be shown on the left in a tree-like view as well as the corresponding weights.

To generate a function that may be used in a program you can press the "Export" button. For each output neuron a file containing the function is written to disk. If more than one output neuron was used these files are numbered according to the position in

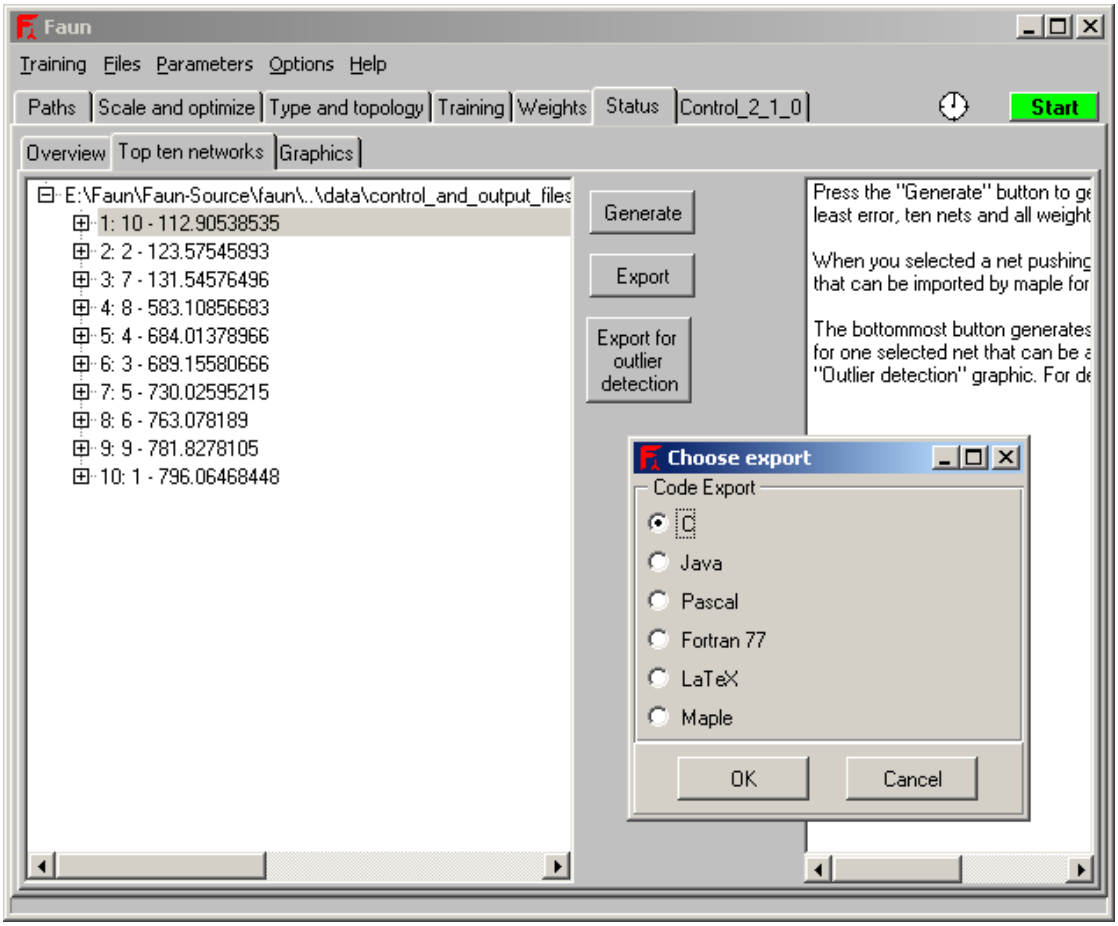


Figure 2.4: "Status"/"Top ten networks" subwindow for a collection of the 10 best neural networks, i. e. the 10 successfully trained networks with the lowest training error $\varepsilon_t(W)$. The analysis is only possible offline after a FAUN run. The "Export" button produces source code to use a successfully trained network in a program. Additionally an input file for Maple can be generated, see figure 2.5.

the network. You can choose from several programming languages to produce source code for or save a textfile (see figure 2.5) as input for Maple to create that function. The following parameters are used for generation so be sure that the corresponding input fields contain correct values:

- Topology setting, i. e. 3- or 4-layered perceptron or radial basis functions network
- Number of neurons of input, hidden and output layers
- Number of parallely trained networks
- Shortcuts on or off

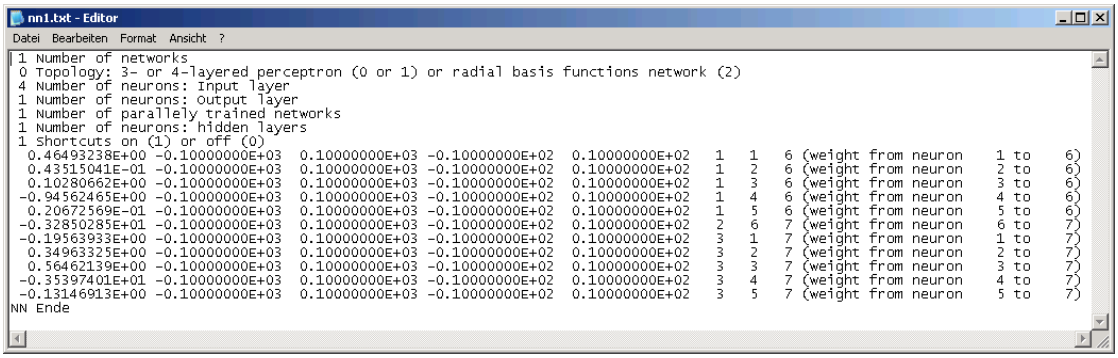


Figure 2.5: Formatted output file `nn1.txt` which can be used easily as input file for a MAPLE-script (see `http://www.maplesoft.com`). The MAPLE-script is a FAUN tool which enables a detailed analysis of trained networks with the computeralgebra system MAPLE. Additionally, among other features, professional 2- and 3-dimensional graphics and an output of runtime optimized C- and FORTRAN-code for the network function are possible with MAPLE.

You may also select a network for outlier detection with the bottommost button. For this to work several parameters will be adjusted *automatically*:

Parameter label	Setting
Number of networks to be trained successfully	1
Maximum number of trainingstops	1
Worst accepted cross-validation quality	1.E20
Number of minimizing iterations without cross-validation	1
Outlier detection	on

A warning appears and allows you to save your data directory (see section 2.1.1) by answering the dialogue with “Yes”. If you don’t need your previous settings or you did already save them then push “No”. A new Faun run will be started immediately. When it has finished, click on the button labeled “Outlier detection” on the “Graphics” tab sheet (see section 2.1.3) to analyze the patterns.

Graphics

With the buttons on this tab sheet you can analyze the quality of your networks. The screenshot in figure 2.6 shows a graphic that was generated after pressing “Errors”: The “Update” button serves as manual update function. When you click the button labeled “Timer off” it will subsequently change to “10 sec”, “30 sec”, “60 sec” and then “Timer off” again. This means every 10 seconds (30 seconds, 60 seconds) the graphic will be updated automatically so you don’t have to press the “Update” button all the time.

The scaling fields let you zoom into the graphic. There is no need to fill in all fields, e.g. if you just want to inspect everything below some value then fill in the field for “y max” with the appropriate value and leave the other fields untouched.

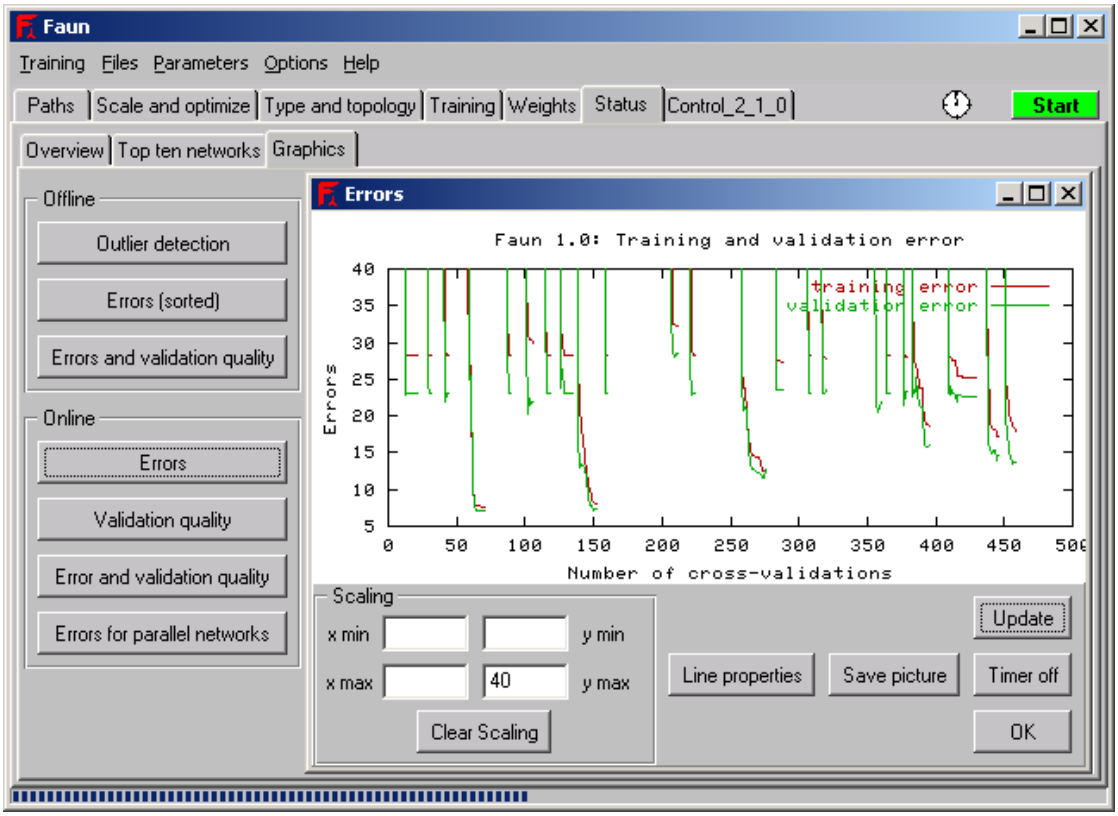


Figure 2.6: "Status"/"Graphics" subwindow for a graphical analysis of the training error and cross-validation error time series. The analysis is possible online during a FAUN run (with automatic update) or offline after a FAUN run. The scaling option facilitates a detailed analysis of single network trainings.

With "Line properties" you can change the color of the lines. However, changing the linewidth is not supported yet.

Clicking on the "Save picture" button saves the graphic as PNG-Picture in the directory you select.

2.2 Menu

2.2.1 Training

Start

Start Faun program.

Before Faun will be started, all parameters are checked for errors. When errors are found, a dialogue will open and show the corresponding tab sheet and an error description (see section 2.2.3). You can leave the window open while checking the parameters.

Now the control files are generated. The are saved in the default directory (see section 2.1.1). Note that the parameters are always taken from the GUI, even if you specified paths for controlfiles. These are only used for loading or saving controlfiles (see section 2.2.2).

If files containing training and validation data are specified, these are copied to the Faun program directory before Faun is started in the background. You can verify your settings by checking the program log on the "Overview" tab sheet which is contained in the "Status" tab sheet.

You can also start Faun with the green button on the top right of the main window which turns red then. When you click the button while it is red it will stop Faun and turn green again to indicate that you can start Faun again now.

A clock to the left of this button shows the elapsed time since start of the last Faun run. To change the clock rate see section 2.2.4.

Stop

Stop Faun program.

This allows you to terminate Faun before it finished, e.g. to adjust some parameters. Alternatively you can press the red button in the upper right corner of the main window. When Faun stopped it will change its color to green.

2.2.2 Files

Load example

This distribution comes with some prepared examples. Click this menu item and the parameters are loaded and training and validation data are copied into the Faun program directory so you can start Faun immediately (see section 2.2.1). Note that this action will overwrite existing training and validation data files in the Faun program directory.

To add an example by yourself you have to edit the files "`DefControl_1_1_0`" and "`DefControl_2_1_0`" in the subdirectory "`examples`". Simply copy and paste an existing example and replace the parameter values. The name of the example as it is shown in the program dialogue is derived from the section names in this files (given in square brackets, e.g. [ExampleTitle]), so be sure to use the same section name in both files. Then copy training and validation data files into the "`examples`" subdirectory. These files need as prefix the example name and an underscore. With an example called "NewExample" these files have to have the filenames (suppose the data is already scaled) "`NewExample_training_scaled`" and "`NewExample_validation_scaled`" respectively.

If you want to use for some reason a different prefix than the name of your example, you can specify your own using the "Prefix" key in the example section (see file "`DefControl_1_1_0`" for example usage). Note that you alone are responsible for assuring that no name conflicts arise.

The space shuttle examples and the analog/digital converter examples have their own controlfiles with the prefix "Shuttle" and "Converter" respectively, so do not use them in your own examples.

All examples are documented in the appendix starting on page 35.

Load directory

This menu item copies the contents of the specified directory into the `data` directory. Make sure that the selected directory has the needed directory structure, i. e. the subdirectories `control_and_output_files` and `data_files`, or ignore the error message and select appropriate paths on the "Paths" tab sheet.

Save directory

Like the previous item copies the contents of a whole directory into the `data` directory this item copies the contents of a whole directory into a directory you select and therefore saves the settings and patterns.

Load controlfiles

This lets you import your own controlfiles as well as program generated ones. Before you can load a controlfile you have to specify the appropriate paths on the "Paths" tab sheet (see section 2.1.1).

Save controlfiles

Once you have found parameters that lead to good results you may want to save them into controlfiles to reuse them later. You have to specify the appropriate paths on the "Paths" tab sheet (see section 2.1.1) before you can save a controlfile. If the directory does not exist, it will be created.

Save output files

Faun generates up to five output files. Here you can specify a directory where all output files with paths as given on the "Paths" tab sheet will be saved (see section 2.1.1). If the directory does not exist, it will be created.

2.2.3 Parameters

Check

This will check all parameters for errors. This is done automatically every time you start Faun.

Do not rely on this feature because it will only find quite obvious errors. Even if no errors are found Faun still might not work as expected or not at all.

Clear all

Clicking on this item resets all parameters.

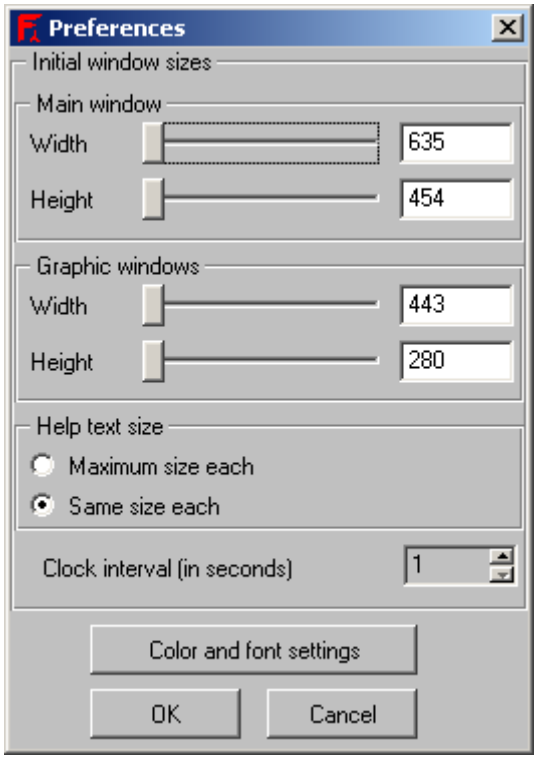


Figure 2.7: Preferences dialogue

2.2.4 Options

Preferences

This opens a dialogue that lets you specify some settings concerning the appearance as shown in figure 2.7.

You may set the initial window sizes for the main window and the graphic windows. The values for the upper bound are determined by your current resolution setting, so you cannot make the windows larger than your screen is. A message will inform you about these bounds when you type into an edit field a value that is out of this range.

If you find the space for displaying help texts too small and you do not want to resize them every time you run the program, you can select "Maximum size each". With this setting the space for help texts is maximized on each tab without hiding any fields and regardless of window size. The default setting "Same size each" reduces this space to the smallest one so that the space for any help text is the same on every tab. You are free to resize these components, however.

Change the clock rate to higher values when you want to track e. g. minutes instead of seconds.

These settings are saved in the file "`configuration.ini`" in the root directory of this distribution.

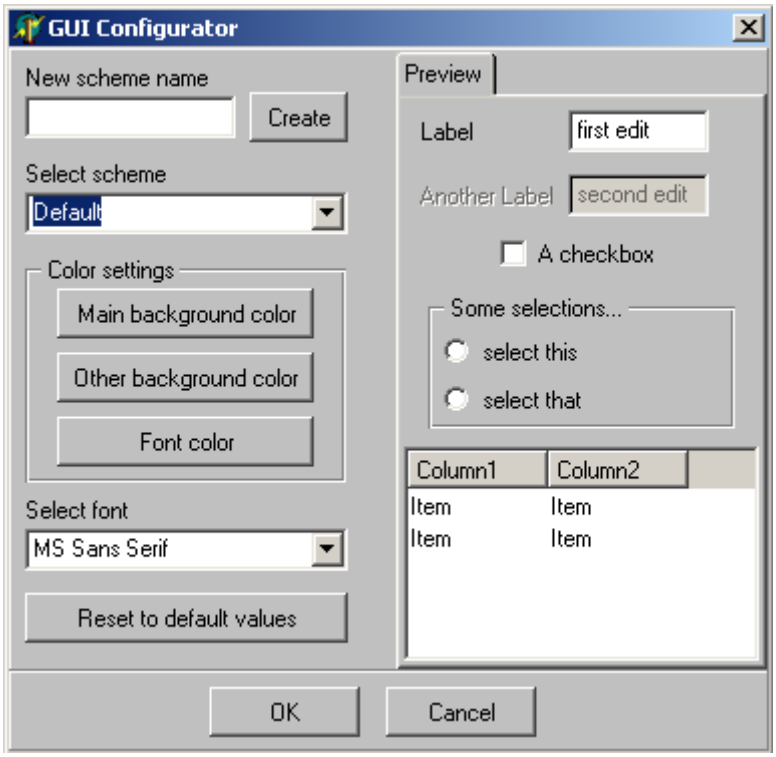


Figure 2.8: Changing color and font settings

Pressing the button labeled "Color and font settings" opens a dialogue that lets you specify a different font and different colors than the default ones (see figure 2.8). All controls will adjust themselves so that with a wider font you can still read all labels. Note that the window size will probably also change.

This window is split into two parts. The right half of the window is used as a preview for the selected settings so you can try different settings without the need to switch back to the main program to evaluate the result.

On the left half you may select your favourite font and colors. Generate a new scheme by entering a name and pressing the create button or choose an existing one from the box below. Settings are applicable only to the selected scheme.

The reset button changes the colors of the selected scheme to the default ones as shown in the screenshot above.

All schemes are saved in the file "`schemes.ini`" in the root directory of this distribution.

2.2.5 Help

Manual

This menu item opens this manual.

Info

Display some information about this project.

2.3 Faun usage without GUI

As Faun is a standalone program it can easily be run without a graphical user interface. Besides the Faun program all you need are four files:

1. `control_1_1_0`
2. `control_2_1_0`
3. `training_scaled` or `training_unscaled`
4. `validation_scaled` or `validation_unscaled`

All parameter settings that are not listed on the tab sheet labeled "Control_2_1_0" are used in the file `control_1_1_0`. For a detailed description of all parameters see section 2.1.2; the given numbers match those in the control file. Letters indicate the first, second, third, ... parameter in one line.

The design of the pattern files is as follows:

- "#" begins a comment until the end of line
- Every pattern consists of two lines: The first line contains the *input* values of every neuron, the second line contains the *output* values of every neuron.

  See the training and validation data files in the `examples` directory for an example.

Put these files into the `data` directory as explained in section 2.1.1 and then start Faun. A window will open in which Faun prints its configuration as set in the controlfiles, the number of training and validation patterns and so on. When starting Faun on a commandline you can redirect this output in a file by typing

```
$> faun > output.log
```

where "output.log" specifies the file that will contain the output. If you want to append these messages to an existing file then type

```
$> faun >> output.log
```

3 Webfaun Documentation

Webfaun is a frontend to FAUN that is developed with PHP (`http://www.php.net`). It was developed by Simon König[1] to utilize compute servers without needing to control FAUN via commandline. An extension for distributed computing is currently under development by Hans-Jörg von Mettenheim.

Usage of the web interface to FAUN is similar to using the standalone GUI described in section 2 on page 5. This section only desribes additional features of the webinterface. As Webfaun is just another frontend to FAUN all parameters discussed in section 2.1.2 on page 5 are also valid for the web interface.

3.1 Faun page

Figure 3.1 shows the main page that is also the first page a user sees after logging in. This pages functionality is grouped into three boxes:

- Managing projects
  A project is the basic management unit of the web interface. It comprises
  - control files,
  - training and validation data,
  - a log of the last FAUN run and
  - graphics that were generated in this context.

  You can create new projects as long as your quota is not exceeded. The "duplicate" action creates a new project with the same files as the selected project, so you can safely experiment with certain settings without loosing previous settings.

  Only one project can be "activated" at any time.

  Downloading a project creates a bzip2[2] compressed tar archive containing all project files.

- Controlling FAUN
  Only one instance of FAUN can be started per project. However, it is possible to create several projects and start FAUN once in each project simultaneously.

  The textarea below shows the log of the actual FAUN run.

- Changing password

[1] `http://www.iwi.uni-hannover.de/mitarbeiter/sk.html`
[2] `http://www.bzip.org`

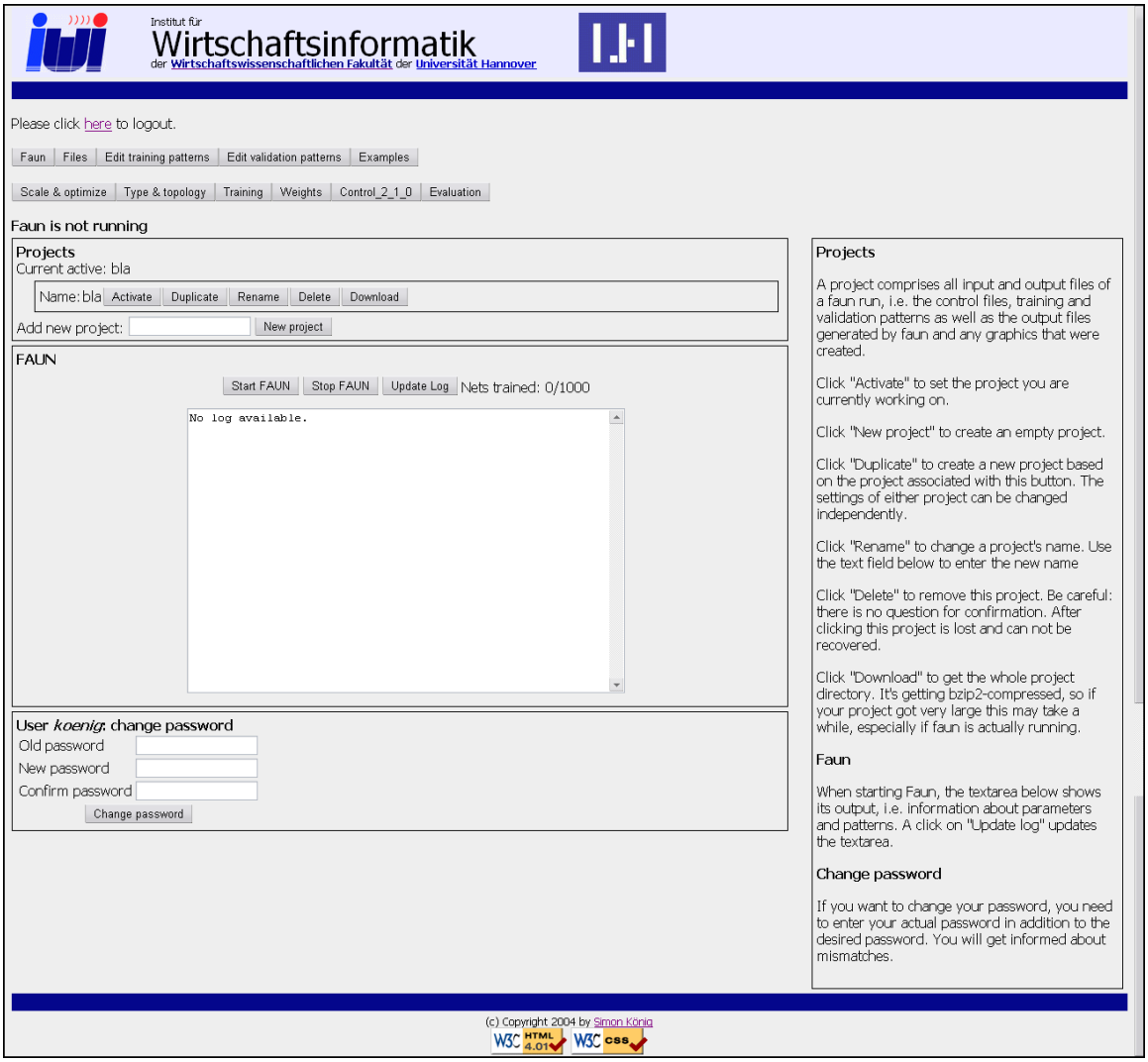


Figure 3.1: “Faun” page of Webfaun

3.2 Files page

The page shown in figure 3.2 allows you to

view The contents of the selected file are displayed in the textarea at the bottom of the page.

download If a file exists the download link is active.

upload Enter a local filename and click "upload" to use a control or ouput file or a file containing patterns.

single files.

This is mainly used for debugging purposes, but can also be utilized for reusing control files, exchanging training or validation data and analysing FAUN generated output files.

3.3 Edit patterns

As shown in figure 3.3 single training and validation patterns can be changed or added without the need to upload complete pattern files. The number of visible patterns per page is customizable.

After changing a pattern *its* update button must be clicked. Otherwise changes are lost.

Training (validation) patterns are only written to disk if you update single neurons or click the "Update" or "Add pattern" buttons at the top of the page, so you can reduce the number of input or output neurons without loosing training and validation patterns at once.

The number of input and output neurons is taken from parameters on the "Training" page (see section 2.1.2).

3.4 Evaluation page

This page provides you with several analysis options. Although these options correspond to their counterparts in the GUI (see sections 2.1.3 and 2.1.3), as export options currently only export to Maple is implemented.

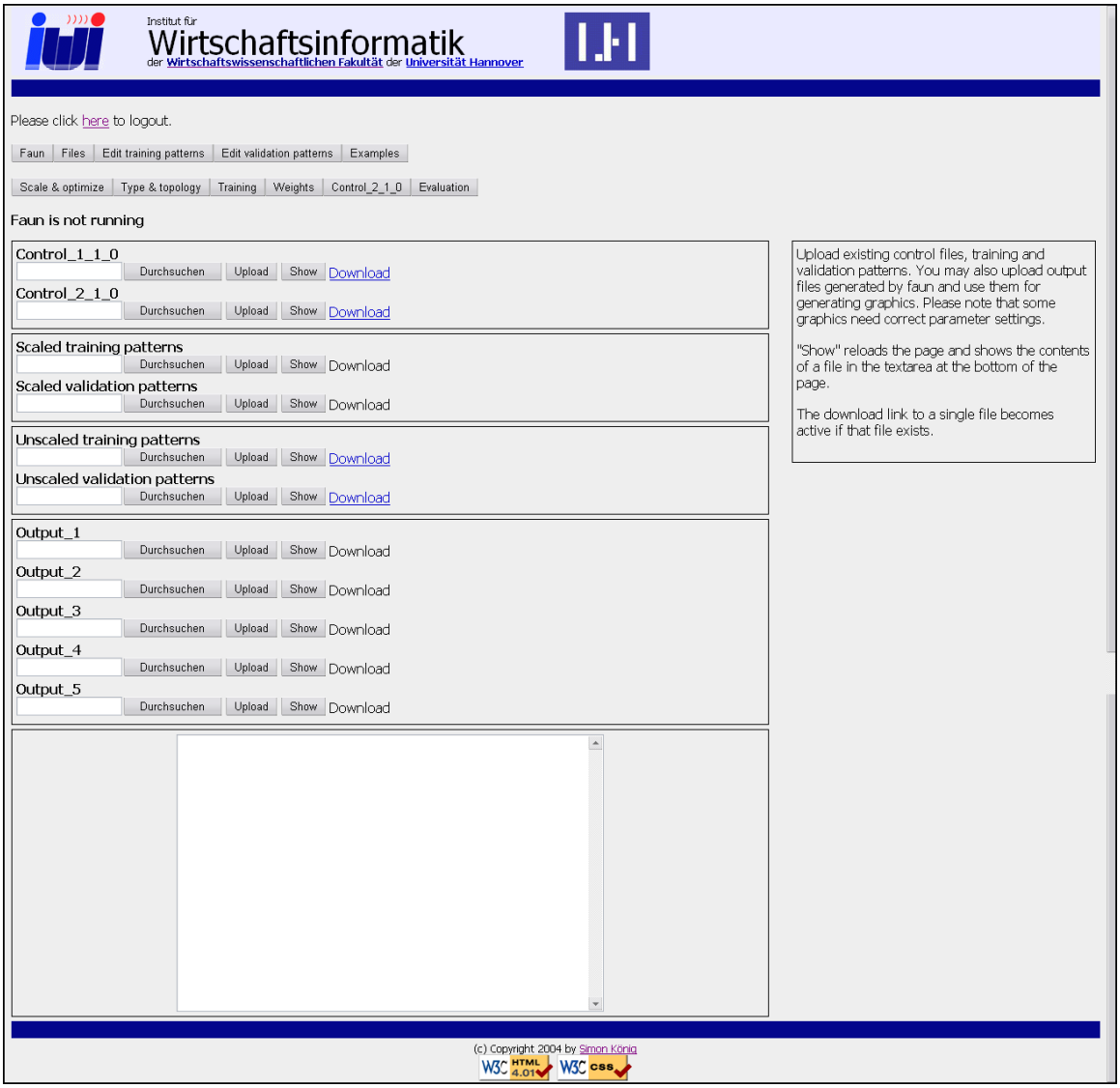


Figure 3.2: “Faun” page of Webfaun

Figure 3.3: "Edit training pattern" page of Webfaun

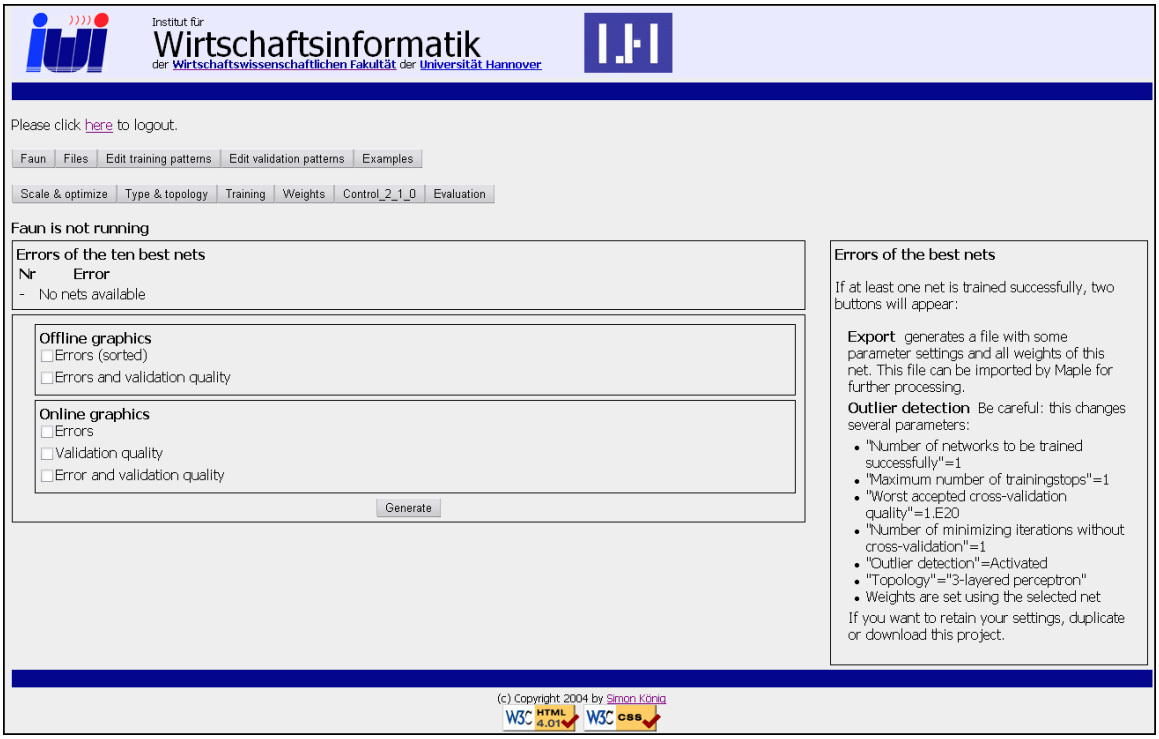


Figure 3.4: "Evaluation" page of Webfaun

Bibliography

[1] Breitner, M. H., and Bartelsen, S., *Training of Large Three-layer Perceptrons with the Neurocomputer SYNAPSE*. Operations Research '98 (selected papers), pp. 562-570, Springer, Berlin 1999.

[2] Breitner, M. H., *Heuristic Option Pricing with Neural Networks and the Neurocomputer SYNAPSE 3*. Optimization 47 (2000), pp. 319-333.

[3] Breitner, M. H., Mehmert, P., and Schnitter, S., *Coarse- and Fine-grained Parallel Computation of Optimal Strategies and Feedback Controls with Multilayered Feedforward Neural Networks*. Proceedings of the Ninth International Symposium on Dynamic Games and Applications in Adelaide, University of South Australia, Adelaide 2000, pp. 107-126.

[4] Breitner, M. H., *Nichtlineare, multivariate Approximation mit Perzeptrons und anderen Funktionen auf verschiedenen Hochleistungsrechnern*, extended and revised Venia Legendi thesis, Akademische Verlagsgesellschaft, Dissertationen zur Künstlichen Intelligenz 263, Berlin 2003 (in German).

[5] Fine, T. L., *Feedforward neural network methodology*, Springer, New York, 1999.

[6] Fletcher, R., *Practical methods of optimization*. John Wiley & Sons, New York, reprint of the 2nd edition, 1999.

[7] Gallant, S. I., *Neural network learning and expert systems*. MIT Press, Cambridge, Massachusetts, 1995.

[8] Gill, P. E., Murray, W., and Saunders, M. A., *Large-scale SQP methods and their Application in Trajectory Optimization*. In Bulirsch, R., and Kraft, D., (eds.), Computational Optimal Control, International Series of Numerical Mathematics 115, Birkhäuser, Basel 1994, pp. 147-162.

[9] Gill, P. E., Murray, W., and Wright, M. H., *Practical Optimization*, 12th Printing, Academic Press, London, England, 1999.

[10] Hertz, J. A., Krogh, A., and Palmer, R. G., *Introduction to the theory of neural computation*. Addison-Wesley, Reading, Massachusetts, 1996.

[11] Müller, B., Reinhardt, J., and Strickland, M. T., *Neural networks*, Springer, Berlin, 1995.

[12] Rojas, R., *Neural networks (A systematic introduction)*. Springer, Berlin, 1996.

[13] Vemuri, V. R., *Artificial neural networks (Concepts and control applications)*. Institute of Electrical and Electronic Engineers (IEEE) Computer Society Press, Los Alamitos, California, 1994.

[14] White, D. A., and Sofge, D. A. (eds.), *Handbook of intelligent control – Neural, fuzzy and adaptive approaches*. Van Nostrand Reinhold, New York, 1992.

[15] White, H., Gallant, A. R., Hornik, K., Stinchcombe, M., and Wooldridge, J., *Artificial neural networks – Approximation and learning theory*. Blackwell, Oxford, 1992.

Appendix

FAUN neurosimulator history

The first version FAUN 0.1, developed between 12/1996 and 2/1998 by Michael H. Breitner, trains 3-layered perceptrons with and without shortcuts. A perceptron's error function and its gradient are computed without matrix algorithms. A preprocessor calculates problem dependent parameters and modifies these parameters in the FORTRAN 77 source code to minimize memory (RAM) allocation. The SQP method NPSOL 5.02 from P. E. Gill, UC San Diego, is implemented. Strategies for the prevention of overtraining are realized, for details see [4].

The updated version FAUN 0.2 has been developed between 3/1998 and 1/1999 by Michael H. Breitner, too. A perceptron's error function and its gradient are computed with matrix algorithms implemented with the BLAS library routines. Online and offline graphic output is realized with GNUPLOT, see `http://www.gnuplot.info`. A statistical evaluation of FAUN and an easy outlier detection are supported. A MAPLE-script enables both a detailed analysis of trained ANNs with the computeralgebra system MAPLE and the output of runtime optimized C- and FORTRAN-code for trained ANNs. The preprocessor has automatic scaling options for the patterns. An interface to the PCI-boards SYNAPSE 2 and SYNAPSE 3 for extremely fast matrix computations is realized, see [1, 2, 4].

FAUN 0.2 was improved to FAUN 0.3 by Patrick Mehmert, Lars Neujahr and Janka Zündel. 4-layered perceptrons with and without shortcuts and radial basis function ANNs are implemented. The SQP method NLSSOL from P. E. Gill, UC San Diego, is added. P. Mehmert developed the PVM environment for FAUN 0.3 between 5/1999 and 7/2000 in cooperation with Stefan Schnitter, formerly Technische Universität Clausthal, and Michael H. Breitner, see [3, 4].

In October 2002 the new FAUN project-group has been launched in Hannover by Michael H. Breitner. Further members of the FAUN project-group are Matthias Kehlenbeck (FORTRAN 95 kernel and compiler options' optimization), Frank Köller (project management, optimization algorithms, source code reengineering, benchmark problems), Roland Kossow (consulting), Hans-Jörg von Mettenheim (FAUN-HPC[3]) and Simon König (pervasive/ubiquitous computing, Delphi[4], Kylix[5] and PHP[6] interfaces). The latest version FAUN 1.1 is now working under Windows 95/98/NT/2000/XP and Linux. It comes with FORTRAN 95 source code with, e. g., dynamic memory allocation and run-time minimized executables, realized by Frank Köller. FAUN has a user-friendly interface developed with Delphi and Kylix, realized by Simon König, and a comprehensive documentation with various examples which enables a user to start immediately. Additionally a webfrontend developed with PHP is available.

The FAUN-HPC family for parallel and array processors as well as homogeneous and inhomogeneous computer clusters and grid computers is still under development.

[3] HPC: High Performance Computing, e. g. PVM- and MPI parallelization.

[4] Delphi: Development environment from Borland to develop Windows applications, see `http://www.borland.com/delphi`.

[5] `http://www.borland.com/kylix`

[6] PHP: Hypertext Preprocessor, see `http://www.php.net`

JOURNAL OF OPTIMIZATION THEORY AND APPLICATIONS: Vol. 107, No. 3, pp. 481–503, DECEMBER 2000

Robust Optimal Onboard Reentry Guidance of a Space Shuttle: Dynamic Game Approach and Guidance Synthesis via Neural Networks[1,2,3]

M. H. Breitner[4]

Communicated by G. Leitmann

Abstract. Robust optimal control problems for dynamic systems must be solved if modeling inaccuracies cannot be avoided and/or unpredictable and unmeasurable influences are present. Here, the return of a future European space shuttle to Earth is considered. Four path constraints have to be obeyed to limit heating, dynamic pressure, load factor, and flight path angle at high velocities. For the air density associated with the aerodynamic forces and the constraints, only an altitude-dependent range can be predicted. The worst-case air density is analyzed via an antagonistic noncooperative two-person dynamic game. A closed-form solution of the game provides a robust optimal guidance scheme against all possible air density fluctuations. The value function solves the Isaacs nonlinear first-order partial differential equation with suitable interior and boundary conditions. The equation is solved with the method of characteristics in the relevant parts of the state space. A bundle of neighboring characteristic trajectories yields a large input/output data set and enables a guidance scheme synthesis with three-layer perceptrons. The difficult and computationally expensive perceptron training is done efficiently with the new SQP-training method

[1]This paper is dedicated to the memory of Professor Rufus Philip Isaacs on the occasion of the 20th anniversary of his death.

[2]This work has been supported in part by the Deutsche Forschungsgemeinschaft, Schwerpunktprogramm "Echtzeit-Optimierung großer Systeme" and Sonderforschungsbereich 255 "Transatmosphärische Flugsysteme". The author gratefully appreciates the help by Professor P. E. Gill, UC San Diego, providing the sequential quadratic programming methods NPSOL and NLSSOL, by Professor R. Bulirsch and Dr. P. Hiltmann, TU München, providing the multiple shooting method MUMUS, and by Dr. Oskar von Stryk, TU München, providing the direct collocation method DIRCOL.

[3]Dynamic games are multistage (multiact) games with a finite or an infinite number of stages (actions of the players). The latter are governed often by ordinary differential equations. Rufus Philip Isaacs, the acknowledged father of dynamic games, used the term "differential games."

[4]Associate Professor for Mathematics and Computer Science, Fachbereich Mathematik und Informatik, Technische Universität Clausthal, Clausthal-Zellerfeld, Germany.

0022-3239/00/1200-0481$18.00/0 © 2000 Plenum Publishing Corporation

FAUN. Simulations show the real-time capability and robustness of the reentry guidance scheme finally chosen.

Key Words. Robust optimal control, Isaacs equation, generalized minimax principle, real-time guidance, neural networks, multivariate approximation, multiple shooting method, space shuttle reentry.

1. Reentry of a European Space Shuttle

A standard manned mission of a future European space shuttle takes place in a circular orbit at an altitude of 250 kilometres and an inclination of 28.5 degrees. At the end of the mission, a controllable deorbit impulse deforms the circle to an ellipse and the shuttle reenters the Earth atmosphere at an altitude of about 95 kilometres; see Fig. 1 for an optimal reentry trajectory computed with nominal air density and neglecting the temperature limit. During the subsequent reentry maneuver, no thrust is available and the annihilation of high total energy and the maneuvering can be done only with aerodynamic forces. The success of a European space shuttle, (i.e., high payload and high reliability) depends strongly on a simultaneous optimization of design and reentry maneuver and on a powerful onboard reentry guidance system; see Refs. 1–7. Due to ionization during the reentry maneuver, all computations must be done in real time and onboard, i.e., on comparably slow space computers. In the sequel, the emphasis is on the

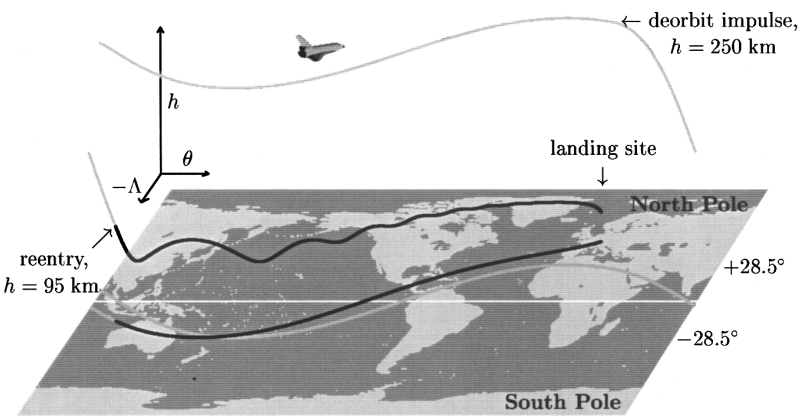


Fig. 1. Return of a future European space shuttle to Earth.

development of a robust optimal real-time guidance scheme for the very critical reentry maneuver.

The shuttle motion can be modeled with sufficient accuracy via a point mass model. The state vector $z := (h, \Lambda, \theta, v, \gamma, \chi)^T \in \mathbb{R}^6$ is defined in spherical coordinates by the position vector $(h, \Lambda, \theta)^T$ and velocity vector $(v, \gamma, \chi)^T$, where h denotes the altitude above the Earth surface, Λ the latitude, θ the longitude, v the velocity, χ the azimuth angle, and γ the flight path angle. The controls for the aerodynamic forces (lift L and drag D) are the lift coefficient C_l and bank angle μ, which must satisfy the technical box constraints

$$0.02 \le C_l \le 0.40, \qquad -\pi/6 \le \mu \le \pi/6. \tag{1}$$

Additionally, four constraints have to be obeyed to limit heating, dynamic pressure, load factor and flight path angle at high velocities. For the air density associated with the aerodynamic forces and the constraints, only an altitude-dependent range can be predicted. Thus, an air density model $\sigma\rho$ based on a nominal air density model $\rho(h)$ [e.g., $\rho(h) = \rho_0 \exp(-\beta h)$] and a deviation factor $\sigma \in [\sigma_{\min}(h), \sigma_{\max}(h)]$ for the unmeasurable and unpredictable density fluctuation is used; see Fig. 2 and Refs. 8–13 for details.

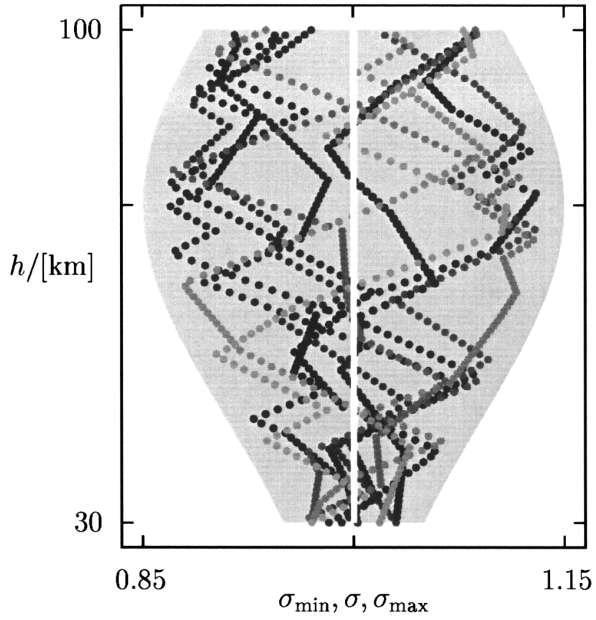


Fig. 2. Air density model $\sigma\rho$ with measurements and fluctuation tube (filled).

The above mentioned control constraints are

$$C_l \le C_{l,\max,h}(h, v, \sigma_{\max}(h); \vartheta_{\max}),$$
$$\text{with temperature limit } \vartheta_{\max}, \tag{2}$$
$$C_l \le C_{l,\max,q}(h, v, \sigma_{\max}(h); q_{\max}),$$
$$\text{with dynamic pressure limit } q_{\max}, \tag{3}$$
$$C_l \le C_{l,\max,l}(h, v, \sigma_{\max}(h); n_{\max}),$$
$$\text{with load factor limit } n_{\max}, \tag{4}$$
$$C_l \le C_{l,\max,f}(h, \Lambda, v, \gamma, \chi, \sigma_{\max}(h); \gamma_{\max}),$$
$$\text{with flight path angle limit } \gamma_{\max}. \tag{5}$$

The control constraint (5) guarantees compliance with the state constraint $\gamma \le \gamma_{\max}$; see Ref. 9. With the control vector $u := (C_l, \mu)^T \in \mathbb{R}^2$ and

$$C_{l,\max} := \min(C_{l,\max,h}, C_{l,\max,q}, C_{l,\max,l}, C_{l,\max,f}, 0.4\}, \tag{6}$$

the set of admissible controls is

$$U(z) := \{u \mid 0.02 \le C_l \le C_{l,\max}(z),\ -\pi/6 \le \mu \le \pi/6\}. \tag{7}$$

To avoid an empty $U(z)$ [i.e., a violation of one or more constraints], the state constraint

$$C_{l,\max}(z) \ge 0.02 \tag{8}$$

must be obeyed too. For the states and trajectories considered, the constraint (8) is monitored and is inactive.

The shuttle motion is governed by the dynamic equations

$$\dot{h} = v \sin\gamma, \tag{9}$$
$$\dot{\Lambda} = (v \cos\gamma \sin\chi)/(R+h), \tag{10}$$
$$\dot{\theta} = (v \cos\gamma \cos\chi)/[(R+h)\cos\Lambda], \tag{11}$$
$$\dot{v} = -[C_{D_l}(h, v) C_l^{1.86} + C_{D_0}(h, v)] S v^2 \rho(h) \sigma/(2m) - g(h) \sin\gamma + \omega^2 (R+h) \cos\Lambda (\sin\gamma \cos\Lambda - \cos\gamma \sin\chi \sin\Lambda), \tag{12}$$
$$\dot{\gamma} = C_l \cos\mu S v \rho(h) \sigma/(2m) - [g(h) \cos\gamma]/v + (v \cos\gamma)/(R+h) + 2\omega \cos\chi \cos\Lambda + \omega^2 \cos\Lambda \times (\sin\gamma \sin\chi \sin\Lambda + \cos\gamma \cos\Lambda)(R+h)/v, \tag{13}$$

$$\dot{\chi} = C_l \sin\mu S v \rho(h) \sigma/(2m \cos\gamma) - v \cos\gamma \cos\chi \tan\Lambda/(R+h) + 2\omega (\sin\chi \cos\Lambda \tan\gamma - \sin\Lambda) - \omega^2 \cos\Lambda \sin\Lambda \cos\chi (R+h)/(v \cos\gamma), \tag{14}$$

where the dot denotes the derivative w.r.t. the independent variable (the time t), R the Earth radius, $C_{D_l}(h, v)$ the induced drag coefficient, $C_{D_0}(h, v)$ the zero-lift drag coefficient, S the aerodynamic reference area, m the mass of the shuttle, $g(h)$ the Earth gravitational acceleration, and ω the Earth angular velocity. The shuttle aerodynamics, mass, and heating at critical stagnation points are taken from an approved and accurate U.S. space shuttle model. All further calculations and computations can be adapted easily to future European space shuttles.

When the shuttle reaches the Earth atmosphere at $t = t_0$, the initial conditions are

$$h(t_0) = 95.000 \text{ km}, \quad \Lambda(t_0) = -0.27394 \text{ rad}, \quad \theta(t_0) = 2.6405 \text{ rad}, \tag{15a}$$
$$v(t_0) = 7.4473 \text{ km s}^{-1}, \quad \gamma(t_0) = -0.021817 \text{ rad}, \quad \chi(t_0) = -0.44534 \text{ rad}. \tag{15b}$$

There holds $\Lambda \in]0, \pi/2]$ on the Northern Hemisphere, $\Lambda \in [-\pi/2, 0[$ on the Southern Hemisphere, $\theta \in [-\pi, 0[$ west of Greenwich, and $\theta \in]0, \pi[$ east of Greenwich. The free terminal time t_f is determined by the condition $v(t_f) = v_f = 1.1160 \text{ km s}^{-1}$. At the velocity v_f, the quasi-steady glide to an airport in Southern Germany (Oberpfaffenhofen, near München) starts. Once the shuttle has reached the Earth atmosphere, both the total energy and velocity decrease continuously and the condition $v(t_f) = v_f$ is certainly fulfilled. The following performance index is minimized:

$$\Phi(t_f) := [(h(t_f) - h_f)/\Delta h_f]^2 + [(\Lambda(t_f) - \Lambda_f)/\Delta\Lambda_f]^2 + [(\theta(t_f) - \theta_f)/\Delta\theta_f]^2 + [(\gamma(t_f) - \gamma_f)/\Delta\gamma_f]^2 + [(\chi(t_f) - \chi_f)/\Delta\chi_f]^2, \tag{16}$$

for

$$h_f = 30.000 \text{ km}, \quad \Delta h_f = 3.000 \text{ km}, \tag{17a}$$
$$\Lambda_f = 0.84038 \text{ rad}, \quad \Delta\Lambda_f = 0.030000 \text{ rad}, \tag{17b}$$
$$\theta_f = 0.19635 \text{ rad}, \quad \Delta\theta_f = 0.045000 \text{ rad}, \tag{17c}$$
$$\gamma_f = -0.047124 \text{ rad}, \quad \Delta\gamma_f = 0.0035000 \text{ rad}, \tag{17d}$$
$$\chi_f = 0.78540 \text{ rad}, \quad \Delta\chi_f = 1.17450 \text{ rad}, \tag{17e}$$

with feedback control vectors $u(z)$ subject to $u(z) \in U(z)$, the constraint (8), and $\sigma \in [\sigma_{\min}(z), \sigma_{\max}(z)]$. Minimization of $\Phi(t_f)$ ensures that the initial conditions for the quasi-steady glide are met with the highest achieveable accuracy against all possible air density fluctuations.

2. Robust Optimal Control

Robust optimal control problems for dynamic systems must be solved if modeling inaccuracies cannot be avoided and/or unpredictable and unmeasurable influences are present. Consider the continuous control process

$$\dot{z} = f(z, u, w), \qquad z(t_0) = z_0, \tag{18}$$

with time t, initial time t_0, state $z \in \mathbb{R}^n$ (w.l.o.g. $z_n \equiv t$ for explicitly time-dependent systems), initial state z_0, control $u \in \mathbb{R}^m$, and unknown modeling inaccuracies and/or unpredictable and unmeasurable influences $w \in \mathbb{R}^p$. Assume that:

u is bounded by $u \in U(z)$, with a closed set $U(z)$ for all $z \in \mathbb{R}^n$, (19)

w is bounded by $w \in W(z)$, with a closed set $W(z)$ for all $z \in \mathbb{R}^n$, (20)

the process terminates after the finite time $t_f - t_0$ with $\Psi(z(t_f)) \leq \epsilon$. (21)

General control and state constraints

$$C(z, u) \leq 0, \qquad C(z, u, w) \leq 0, \qquad C(z, w) \leq 0, \qquad \text{or } C(z) \leq 0$$

can be usually satisfied with control constraints of type (19); see Ref. 9. A feedback control $u(z)$ is admissible if and only if condition (19) holds and $u(z)$ guarantees termination after a finite time; see condition (21), for all $w \in W(z)$. All states $z \in \mathbb{R}^n$ for which an admissible feedback control $u(z)$ exists generate the set S_c of controllable states. For a given performance index $\Phi(z(t_f)) \colon \mathbb{R}^n \to \mathbb{R}$, an admissible feedback control $u^*(z)$ is robust optimal if and only if $u^*(z)$ minimizes $\Phi(z(t_f))$ for all $z_0 \in S_c$ against all $w \in W(z)$. For all $z_0 \in S_c$, the guaranteed value $V(z_0)$ of the performance index is defined by $V(z_0) = \Phi(z(t_f))$ using a robust optimal feedback control $u^*(z)$.

Since the bounds are known for the unknown w [see condition (20)], a worst-case analysis with an antagonistic noncooperative two-person dynamic game is possible; see Refs. 14–18 and also Refs. 8–13. A closed-form solution of this game provides a robust optimal feedback control $u^*(z)$ against all $w \in W(z)$. For the control process (18)–(21), $V(z)$ solves the generally nonlinear first-order partial differential equation (Isaacs equation)

$$V_z \cdot f(z, u^*(V_z, z), w^-(V_z, z, u^*(V_z, z))) \equiv 0, \qquad \text{for all } z \in S_c, \tag{22}$$

with $V_z = \partial V / \partial z$ and (generalized Isaacs' minimax principle)

$$u^*(V_z, z) = \arg \min_{u \in U(z)} V_z \cdot f(z, u, w^-(V_z, z, u)), \tag{23}$$

$$w^-(V_z, z, u) = \arg \max_{w \in W(z)} V_z \cdot f(z, u, w), \tag{24}$$

with suitable interior conditions (singular hypermanifolds, V and V_z discontinuities possible)

$$\Theta_i(V_z^-, V_z^+, z) = 0, \qquad \text{at } \Gamma_i(V_z, z) = 0, \qquad i = 1, 2, 3, \ldots, \tag{25}$$

$\Theta_i \colon \mathbb{R}^{3n} \to \mathbb{R}^n$, $\Gamma_i \colon \mathbb{R}^{2n} \to \mathbb{R}$, and suitable boundary conditions (transversality conditions)

$$\Theta_b(V_z, z) = 0, \qquad \text{for } z(t_f) \in S_c, \tag{26}$$

$\Theta_b \colon \mathbb{R}^{2n} \to \mathbb{R}^n$. Note that very difficult singular manifolds of codimension two or higher are possible, but they are unusual in the relevant parts of the state space.

The partial differential equation (22)–(26) is solvable with the method of characteristics applicable to all partial differential equations of first order. Every characteristic trajectory $(z^*(t)^T, V_z(z^*(t)), V(z^*(t)))^T$, $t \in [t_0, t_f]$, is solution of a multipoint boundary-value problem with $2n+1$ ordinary differential equations. The system of ordinary differential equations has the independent variable t and the dependent variables z, V_z, V. In general, the multipoint boundary-value problem can be solved only numerically by means of very sophisticated multiple shooting or collocation methods; see Refs. 19–21. By homotopy techniques, the relevant parts of S_c can be filled often with many characteristic trajectories (i.e., 100, 1000, 10,000, 100,000, ...) depending on the problem considered.

The large input/output data set $(z_k, V_z(z_k))$, $k = 1, 2, \ldots$, obtained along the trajectories enables a synthesis of $V_z(z)$ for $z \in S_c$. Two synthesis approaches turned out to be successful for real-life (robust) optimal control problems: (i) many Taylor series expansions with global smoothing; see Refs. 9, 11, 12; and (ii) supervised training of artificial neural networks; see Refs. 9, 18, 22. For the second approach, three-layer and four-layer perceptrons (1 to 20 hidden neurons) for each component of $V_z(z)$ have been appropriate for the applications investigated. For real-life applications (i.e., with a large state vector of dimension n and/or with large input/output data sets), artificial neural networks are computationally expensive.

In the sequel, the emphasis is on the efficient training of three-layer perceptrons to synthesize a robust optimal real-time guidance scheme $u^*(z, V_z(z))$ according to Eq. (23) for the above mentioned reentry maneuver.

3. Synthesis of a Robust Optimal Real-Time Guidance Scheme

To generate the large input/output data set $(z_k, V_z(z_k))$, $k = 1, 2, \ldots$, a bundle of neighboring characteristic trajectories with slightly-varied initial

conditions (15) must be computed. The difficulties in applying the indirect multiple shooting method are caused by the a priori unknown switching structure (i.e., by the occurrence of singular hypersurfaces) and by the required sufficiently accurate initial guess for $(z^*(t)^T,\ V_z(z^*(t)),\ V(z^*(t)))^T$, $t \in [t_0, t_f]$.

Briefly outlined, the following new approach for the computation of the characteristic trajectories for complex dynamic games has been used; see Refs. 9–10 for details. In a first step, the dynamic game is simplified tightening the possible air density fluctuations to $\sigma \equiv 1$ [i.e., using the nominal air density $\rho(h)$] and neglecting the temperature constraint $\vartheta \leq \vartheta_{\max}$. This leads to a much easier solvable optimal control problem. By discretization of the state and control variables, the infinite-dimensional optimal control problem can be transformed into an optimization problem in a finite-dimensional space. The direct collocation method DIRCOL (see Refs. 23–24) is used to optimize the entry point into the Earth atmosphere in order to decrease the maximum bank angle, maximum dynamic pressure, and maximum aerodynamic heating; see Figs. 1, 3, 4 and Refs. 5, 9. For $\sigma \equiv 1$, DIRCOL provides an accurate estimate for the characteristic trajectory $(\bar{z}^*(t)^T,\ \bar{V}_z(\bar{z}^*(t)),\ \bar{V}(\bar{z}^*(t)))^T$, $t \in [t_0, \bar{t}_f]$. Thus, an excellent initial guess

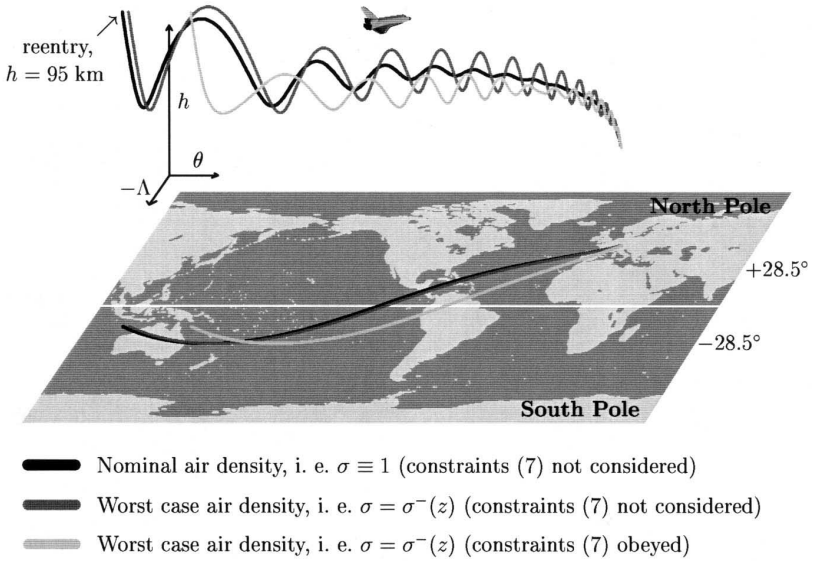


Fig. 3. Homotopy of characteristic trajectories.

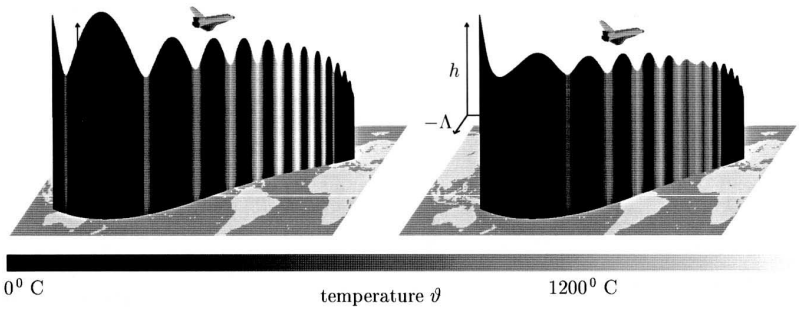


Fig. 4. Characteristic trajectories against worst-case air density.

for the subsequent solution of the related multipoint boundary-value problem by the indirect multiple shooting method MUMUS[5] is available (second step).

In a third step, the tightened air density fluctuations are slowly unchained by homotopy, i.e., by

$$\sigma \in [1 - \alpha_1 + \alpha_1 \sigma_{\min}(h),\ 1 - \alpha_1 + \alpha_1 \sigma_{\max}(h)],$$

with homotopy parameter $\alpha_1 \in [0, 1]$. Analogously, the maximum temperature constraint (2) can be recovered by

$$C_l \leq 0.4(1 - \alpha_2) + \alpha_2 C_{l,\max,h}(h, v, \sigma_{\max}(h); \vartheta_{\max}),$$

with homotopy parameter $\alpha_2 \in [0, 1]$; see Figs. 3–4. Characteristic reentry trajectories against the worst-case air density [i.e., $\sigma = \sigma^-(z)$], for an optimized entry point into the Earth atmosphere are compared in Fig. 4. Along the left trajectory, some of the constraints (7) are not considered and the maximum temperature is 1700° Celsius. Along the right trajectory, the constraints (7) are obeyed and the maximum temperature is 1200° Celsius.

In a fourth step, the initial conditions (15) are slightly varied to compute a bundle of 1000 neighboring characteristic trajectories. Although the switching structure changes, the characteristic trajectories can be computed very efficiently, i.e., fast and semiautomatically. Taylor series expansions of second order for the components $V_v(z)$, $V_\gamma(z)$, $V_\chi(z)$ of $V_z(z)$ are fit at 308 approximation points within the bundle of trajectories (last step). The Taylor series expansions enlarge significantly the part of the state space where data $(z_k, V_z(z_k))$ are available. The drawback is that these data are no longer free of noise.

[5]MUMUS: Munich multiple shooting method.

With $\cos\gamma > 0$ and $V_v(z) < 0$, for all $z \in S_c$, the robust optimal shuttle guidance scheme $u^*(z, V_z(z))$ [see Eq. (23)] reads as follows:

$$\begin{aligned}
&\sin\mu_f^*(z, V_z(z)) \\
&= -V_\chi(z)/(\cos\gamma\sqrt{[V_\chi(z)/\cos\gamma]^2 + V_\gamma(z)^2}) \\
&= \operatorname{sign}[V_\chi(z)/V_v(z)]/\sqrt{1 + \{[\cos\gamma\, V_\gamma(z)/V_v(z)]/[V_\chi(z)/V_v(z)]\}^2}, \qquad (27)
\end{aligned}$$

$$\begin{aligned}
&\mu^*(z, V_\chi(z)/V_v(z), V_\gamma(z)/V_v(z)) \\
&= \begin{cases} -\pi/6 & \text{if } \sin\mu_f^* < -\sin(\pi/6), \\ \arcsin(\sin\mu_f^*), & \text{if } |\sin\mu_f^*| \le \sin(\pi/6), \\ \pi/6, & \text{if } \sin\mu_f^* > \sin(\pi/6), \end{cases} \qquad (28)
\end{aligned}$$

and moreover,

$$\begin{aligned}
C_{lf}^*(z, V_z(z)) = {} & \{[V_\chi(z)/V_v(z)]\sin\mu^*(z, V_z(z))\}/[1.86 C_{D_l}(z) v\cos\gamma] \\
& + [V_\gamma(z)/V_v(z)]\cos\mu^*(z, V_z(z))/[1.86 C_{D_l}(z) v]^{1/0.86}, \qquad (29)
\end{aligned}$$

$$\begin{aligned}
&C_l^*(z, V_\chi(z)/V_v(z), V_\gamma(z)/V_v(z)) \\
&= \begin{cases} 0.02, & \text{if } C_{lf}^* < 0.02, \\ C_{lf}^*, & \text{if } 0.02 \le C_{lf}^* \le C_{l,\max}(z), \\ C_{l,\max}(z), & \text{if } C_{lf}^* > C_{l,\max}(z). \end{cases} \qquad (30)
\end{aligned}$$

Instead of $V_z(z) \in \mathbb{R}^6$, only $V_\chi(z)/V_v(z)$ and $V_\gamma(z)/V_v(z)$ must be synthesized. Scaling and equilibration of the data set $(z_k, V_\chi(z_k)/V_v(z_k))$, $k = 1, 2, \ldots, n_p$, with $n_p = 13269$,

$$x_{k,i} := 2\left[z_{k,i} - \min_{l=1,\ldots,n_p} z_{l,i}\right] \Big/ \left[\max_{l=1,\ldots,n_p} z_{l,i} - \min_{l=1,\ldots,n_p} z_{l,i}\right] - 1, \quad i = 1, 2, \ldots, 6, \qquad (31)$$

$$\begin{aligned}
y_k := {} & 1.9\left[V_\chi(z_k)/V_v(z_k) - \min_{l=1,\ldots,n_p} V_\chi(z_l)/V_v(z_l)\right] \Big/ \\
& \left[\max_{l=1,\ldots,n_p} V_\chi(z_l)/V_v(z_l) - \min_{l=1,\ldots,n_p} V_\chi(z_l)/V_v(z_l)\right] - 0.95, \qquad (32)
\end{aligned}$$

yields an appropriate pattern set

$$P := \{(x_1, y_1), (x_2, y_2), \ldots, (x_{n_p}, y_{n_p})\}.$$

The pattern set P is divided into the training set

$$P_t := \{(x_k, y_k) | k \in I_t\},$$

with

$$\text{index set } I_t := \{i_{t,1}, i_{t,2}, \ldots, i_{t,n_t}\}, \qquad \operatorname{card}(I_t) = n_t = 8865,$$

and the cross-validation set

$$P_v := \{(x_k, y_k) | k \in I_v\},$$

with

$$\text{index set } I_v := \{i_{v,1}, i_{v,2}, \ldots, i_{v,n_v}\}, \qquad \operatorname{card}(I_v) = n_v = 4404.$$

There holds

$$P_t \cup P_v = P, \qquad P_t \cap P_v = \varnothing.$$

Analogously, a training set with 8981 patterns and a cross-validation set with 4433 patterns is generated for the data set $(z_k, V_\gamma(z_k)/V_v(z_k))$, $k = 1, 2, \ldots, 13414$.

Topologically appropriate neural networks are given by three-layer perceptrons with shortcuts (direct input and output connection), with auxiliary neuron N_0 (bias neuron) with $x_{k,0} := 1$, with input neurons $N_1 - N_6$, with hidden neurons $N_7 - N_{6+n_h}$, $1 \le n_h \le 10$, with output neuron N_{7+n_h}, and with the hyperbolic tangent transfer function,

$$\begin{aligned}
&\text{on}_{k,i} := x_{k,i}, && \text{for } i = 0, 1, \ldots, 6, \\
&\text{on}_{k,i} := \tanh\left(\sum_{j=0}^{6} w_{j,i}\,\text{on}_{k,j}\right), && \text{for } i = 7, \ldots, 6 + n_h, \\
&\text{on}_{k,7+n_h} := \tanh\left(\sum_{j=0}^{6+n_h} w_{j,7+n_h}\,\text{on}_{k,j}\right), && \text{with } k \in I_t \cup I_v.
\end{aligned}$$

N_i's output for the pattern (x_k, y_k) is denoted by $\text{on}_{k,i}$. Generally, only one output neuron is advisable [i.e., separate perceptrons for the output $V_\chi(z)/V_v(z)$ and the output $V_\gamma(z)/V_v(z)$] to facilitate a perceptron training. All weights $w_{j,i}$ of the weight matrices

$$W_{12} = \begin{bmatrix} w_{0,7} & \cdots & w_{6,7} \\ \vdots & \ddots & \vdots \\ w_{0,6+n_h} & \cdots & w_{6,6+n_h} \end{bmatrix}, \qquad W_{12} \in \mathbb{R}^{n_h,7}, \text{ layer } 1 \to \text{layer } 2,$$

$$W_{23} = [w_{7,7+n_h}, \ldots, w_{6+n_h,7+n_h}], \qquad W_{23} \in \mathbb{R}^{1,n_h}, \text{ layer } 2 \to \text{layer } 3,$$

$$W_{13} = [w_{0,7+n_h}, \ldots, w_{6,7+n_h}], \qquad W_{13} \in \mathbb{R}^{1,7} \text{ (shortcuts)},$$

are trainable. With $W := (W_{12}, W_{23}, W_{13})$, the approximation quality of a perceptron can be estimated with the error functions

$$\epsilon_t(W) := (1/2) \sum_{k \in I_t} [\mathrm{on}_{k,7+n_h}(W) - y_k]^2, \tag{33}$$

$$\epsilon_v(W) := \mathrm{card}(I_t)/\mathrm{card}(I_v)(1/2) \sum_{k \in I_v} [\mathrm{on}_{k,7+n_h}(W) - y_k]^2. \tag{34}$$

A perceptron is trained iteratively, i.e., ϵ_t is decreased by adaption of W until ϵ_v increases for two consecutive iterations (prevention of overtraining). Note that the training stops before a local minimum of ϵ_t is reached. Weight upgrades $W_{\mathrm{new}} - W_{\mathrm{old}}$ can be calculated with any minimization algorithm, e.g., with the usually favored first-derivative methods such as the steepest descent or with second derivative methods such as the Newton method. For first-derivative methods, we have the iterative sequence

$$W_{\mathrm{new}} = W_{\mathrm{old}} + \eta \left(\epsilon_t(W_{\mathrm{old}}), \operatorname*{grad}_W \epsilon_t(W_{\mathrm{old}}) \right) \Delta W \left(\epsilon_t(W_{\mathrm{old}}), \operatorname*{grad}_W \epsilon_t(W_{\mathrm{old}}) \right), \tag{35}$$

with search direction ΔW and steplength η. Approved numerical methods for constrained nonlinear least-squares problems [see Eq. (33)] are sequential quadratic programming (SQP) methods and generalized Gauss–Newton (GGN) methods, which can exploit the special structure of the Hessian matrix of ϵ_t; see Refs. 25–28. SQP and GGN methods can overcome automatically most of the training problems of perceptrons, e.g., flat spots or steep canyons of the error function ϵ_t. But SQP and GGN methods converge only toward one of the usually many local minima of ϵ_t. The more neural networks with different weight initializations are trained, the more likely it is to train a perceptron successfully, i.e., to find a neighborhood of a good local minimum. SQP and GGN methods usually approximate the Hessian matrix of ϵ_t by finite differences and update formulas, i.e., become first derivative methods, and can deal with box constraints, linear constraints, and smooth nonlinear constraints. Advantages of these methods are:

(i) A much better search direction ΔW is calculated in comparison to common training methods, e.g. $\Delta W := \mathrm{grad}_W \epsilon_t$ for the gradient method (backpropagation).

(ii) The steplength η is optimized permanently in contrast to common training methods with fixed steplength. Thus, the number of learning steps is reduced significantly by a factor 10, 100, 1000, ...

(iii) Only ϵ_t, $\mathrm{grad}_W \epsilon_t$, and ϵ_v are required, which can be computed mainly by very fast matrix operations. For other neural network topologies (e.g., radial basis functions), an efficient code for $\mathrm{grad}_W \epsilon_t$ also can be deduced by automatic differentiation; see Ref. 30.

(iv) Maximum and minimum of each weight can be set easily (box constraints).

(v) The total curvature of the neural network can be constrained (prevention from neural network oscillations).

(vi) Convexity and monotonicity constraints can be set.

The neural network training method FAUN 0.2 developed by the author[6] is used to train three-layer V_χ / V_v-perceptrons and three-layer V_γ / V-perceptrons with shortcuts. FAUN 0.2 incorporates the SQP methods NPSOL and NLSSOL developed by P. E. Gill, University of California at San Diego.[7,8] A priori investigations omitted show that trained V_χ / V_v-perceptrons and V_γ / V_v-perceptrons with $\epsilon_v > 1.15\epsilon_t$ most likely are overtrained. Thus, the condition

$$\epsilon_v \leq 1.15\epsilon_t \qquad \text{(overtraining limit)} \tag{36}$$

is used to decide whether a trained perceptron is successfully trained or overtrained. With FAUN 0.2 for each topology, $n_h = 1, 2, \ldots, 10$, V_χ / V_v-perceptrons and V_γ / V_v-perceptrons with randomly initialized weights $w_{j,i}$ are trained until $100 V_\chi / V_v$-perceptrons and $100 V_\gamma / V_v$-perceptrons are successfully trained; see Table 1 and Fig. 5 for the training error ϵ_t and cross-validation error ϵ_v of the successfully trained and reasonable perceptrons. Scaled and equilibrated pattern sets [see Eqs. (31)–(32)] for the functions V_χ / V_v (left) and V_γ / V_v (right) are used. In the upper left of Fig. 5 are the regions where perceptrons are most likely overtrained.

Shortly summarized, the FAUN 0.2 computations show that:

(i) Between 796 and 1900 and between 384 and 605 weight initializations and training runs are necessary to train 100 perceptrons successfully; see rows 5 and 12 in Table 1.

(ii) Best training error ϵ_t^* and corresponding cross-validation error ϵ_v^* decrease only slightly with increasing the complexity of the perceptrons topology (i.e., with increasing the number of trainable weights); see rows 3, 6, 13 in Table 1.

[6] FAUN 0.2: Fast approximation with universal neural networks (version 0.2). Also, Faun (Faunus): A woodland diety, later identified with Pan, represented as a man with the ears, horns, tail, and the legs of a goat.

[7] NPSOL: Nonlinear programming problem solver.

[8] NLSSOL: Nonlinear least squares problem solver.

Table 1. SQP training performance of FAUN 0.2.

Topology	I	II	III	V	X
Number n_h of hidden neurons	1	2	3	5	10
Number of trainable weights	9	17	25	41	81
Perceptrons for $V_\chi(z)/V_v(z)$					
Number of weight initializations	1900	1242	931	844	845
Minimum error ϵ_t^* after successful training	20.39	17.27	17.24	16.64	15.66
Percentage with $\epsilon_t \in [\epsilon_t^*, 39]$	62%	58%	61%	43%	20%
Average training time [sec]	131	164	195	282	1038
Percentage of time for ϵ_t evaluation	66%	62%	62%	67%	57%
Percentage of time for $\text{grad}_W \epsilon_t$ evaluation	31%	36%	36%	30%	41%
Perceptrons for $V_\gamma(z)/V_v(z)$					
Number of weight initializations	556	605	464	384	413
Minimum error ϵ_t^* after successful training	28.65	27.64	26.54	26.52	29.56
Percentage with $\epsilon_t \in [\epsilon_t^*, 49]$	26%	29%	26%	18%	2%
Average training time [sec]	38	72	79	105	293
Percentage of time for ϵ_t evaluation	69%	67%	67%	71%	69%
Percentage of time for $\text{grad}_W \epsilon_t$ evaluation	29%	30%	31%	26%	29%

(iii) The percentage of successfully trained perceptrons with reasonable ϵ_t decreases with increasing the complexity of the perceptrons topology; see rows 3, 7, 14 in Table 1.

(iv) The average computing time per successfully trained perceptron increases considerably with increasing the complexity of the perceptrons topology; see rows 3, 8, 15 in Table 1.

(v) About 65% of the computing time is used to evaluate ϵ_t, only about 2% of the computing time is used to evaluate ϵ_v, and about 32% of the computing time is used to evaluate $\text{grad}_W \epsilon_t$; see rows 9, 10, 16, 17 in Table 1.

(vi) About 45% of the computing time is used in the BLAS subroutine DGEMM; see Ref. 8.

(vii) The overtraining limit (36) is more restrictive for the training of the V_χ/V_v-perceptrons; see rows 5, 12 in Table 1. More reasonable V_χ/V_v-perceptrons than V_γ/V_v-perceptrons are among the successfully trained perceptrons; see rows 7, 14 in Table 1 and Fig. 5.

For details see Ref. 8.

FAUN 0.2 computes ϵ_t, ϵ_v, and $\text{grad}_W \epsilon_t$ (99% of the computing time, see rows 9, 10, 16, 17 in Table 1) by accelerable and parallelizable matrix operations. With

$$A := [x_{i_{t,1}}, x_{i_{t,2}}, \ldots, x_{i_{t,n_t}}] \in \mathbb{R}^{7,n_t}, \qquad Y := [y_{i_{t,1}}, y_{i_{t,2}}, \ldots, y_{i_{t,n_t}}] \in \mathbb{R}^{1,n_t},$$

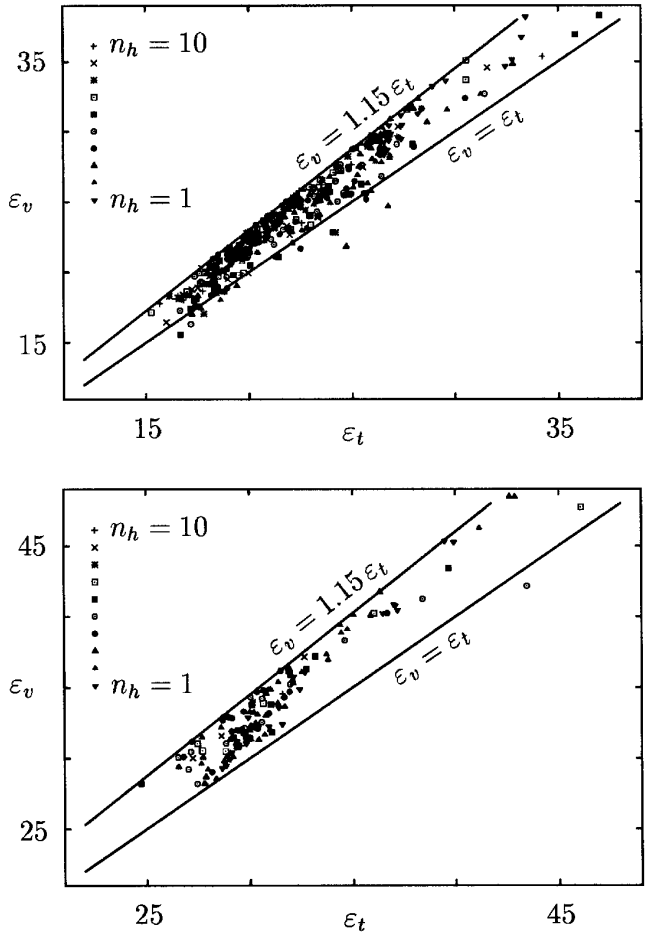


Fig. 5. Training error ϵ_t and cross-validation error ϵ_v.

the training error ϵ_t and its gradient $\text{grad}_W \epsilon_t$ are computable via

$$A_2 := W_{12} A_1, \qquad A_2 \in \mathbb{R}^{n_h,n_t}, \tag{37}$$

$$A_2 := \tanh(A_2), \tag{38}$$

$$A_3 := W_{23} A_2 + W_{13} A_1, \qquad A_3 \in \mathbb{R}^{1,n_t}, \tag{39}$$

$$A_3 := \tanh(A_3), \tag{40}$$

$$S_3 := A_3 - Y, \qquad S_3 \in \mathbb{R}^{1,n_t} \tag{41}$$

$$\epsilon_t(W) := (1/2) S_3 \odot S_3 = (1/2) S_3 S_3^T, \tag{42}$$

and

$$D_3 := \tanh'(\tanh^{-1}(A_3)), \qquad D_3 \in \mathbb{R}^{1,n_t}, \tag{43}$$

$$E_3 := S_3 \odot D_3, \qquad E_3 \in \mathbb{R}^{1,n_t}, \tag{44}$$

$$S_2 := W_{23}^T E_3, \qquad S_2 \in \mathbb{R}^{n_h,n_t}, \tag{45}$$

$$D_2 := \tanh'(\tanh^{-1}(A_2)), \qquad D_2 \in \mathbb{R}^{n_h,n_t}, \tag{46}$$

$$E_2 := S_2 \odot D_2, \qquad E_2 \in \mathbb{R}^{n_h,n_t}, \tag{47}$$

$$G_{12} := E_2 A_1^T, \qquad G_{12} \in \mathbb{R}^{n_h,7}, \tag{48}$$

$$G_{23} := E_3 A_2^T, \qquad G_{23} \in \mathbb{R}^{1,n_h}, \tag{49}$$

$$G_{13} := E_3 A_1^T, \qquad G_{13} \in \mathbb{R}^{1,7}, \tag{50}$$

$$\operatorname*{grad}_W \epsilon_t(W) := [G_{12}, G_{23}, G_{13}], \tag{51}$$

where $\odot$, $\tanh(\ldots)$, $\tanh^{-1}(\ldots)$, and $\tanh'(\ldots)$ denote the elementary matrix multiplication, the elementary hyperbolic tangent function, its elementary inverse function, and its elementary derivative, respectively. With

$$A_1 := (x_{i_{v,1}}, x_{i_{v,2}}, \ldots, x_{i_{v,n_v}}) \in \mathbb{R}^{7,n_v}, \qquad Y := (y_{i_{v,1}}, y_{i_{v,2}}, \ldots, y_{i_{v,n_v}}) \in \mathbb{R}^{1,n},$$

the cross-validation error ϵ_v is computable analogously according to Eqs. (37)–(42). The matrix operations in Eqs. (37)–(50) are implemented with the public domain[9] FORTRAN BLAS level 2 and level 3 (see Ref. 31); e.g., the matrix operations in Eqs. (37), (39), (41), (45), (48)–(50) are implemented with the subroutine DGEMM.[10] The BLAS level 1, level 2, and level 3 are available via NetLib; see http://www.netlib.org. The BLAS is among the most widely-used standard mathematical subroutine libraries in the world. Optimized and parallelized implementations of the BLAS exist for different hardware platforms and both UNIX and WINDOWS systems.

A SUN Enterprise 3500 with four UltraSPARC 336 MHz processors (EP 3500/4/336 for short) and with the commercial SUN Performance Library is used; see http://www.sun.com. The Performance Library makes available optimized and parallelized BLAS levels 1–3 and optimized standard function calls (e.g., for the hyperbolic tangent function). The combination of high performance and use of widely accepted standards means

[9]BLAS: Basic linear algebra subprograms.

[10]DGEMM: Double precision matrix-matrix multiplication on general matrices.

that Performance Library increases the speed of FAUN 0.2 up to a factor of two (single processor) without requiring any source code changes. Because the parallelization is built in an additional speedup on the EP 3500/4/336, it is possible without the expense and effort of modifying FAUN 0.2. The SQP Methods NPSOL and NLSSOL are based on BLAS level 2 and level 3; they are automatically optimized and parallelized too. Note that parallel computing with 2, 3, or 4 processors is advisable only for the retraining of good perceptrons with additional patterns. The training of new perceptrons is more efficient with 2, 3, or 4 jobs which use only a single processor. A comparison of computing times including a widely used single 400 MHz Intel Pentium II processor with 100 MHz bus running on LINUX can be found in Table 2; see Ref. 8 for details.

Available commercial or public domain optimized BLAS for Pentium II processors running on LINUX or WINDOWS are not used; see e.g. http://www.cs.utk.edu/~ghenry/distrib/.

Acceptable computing times can be achieved also by inexpensive specialized neurocomputers which support parallel or matrix algorithms in hardware and speed up compute power up to two orders of magnitude. Ongoing research in the author group is focused on the newly developed SYNAPSE 3 neuro-computer[11] PCI-board based on the patented MA16 neuro-signal

Table 2. Comparison of scalar and parallel FAUN 0.2 computing times.

Topology	I	II	III	V	X
n_h	1	2	3	5	10
Perceptrons for $V_\chi(z)/V_v(z)$					
EP 3500 average training time [sec], 1 processor	131	164	195	282	1038
Pentium II average training time [sec]	285	349	341	678	1303
Speedup	0.46	0.47	0.57	0.42	0.80
Perceptrons for $V_\gamma(z)/V_v(z)$					
EP 3500 average training time [sec], 1 processor	38	72	79	105	293
EP 3500 average training time [sec], 2 processors	35	49	68	107	170
Speedup	1.09	1.47	1.16	0.98	1.72
EP 3500 average training time [sec], 3 processors	32	49	60	80	140
Speedup	1.19	1.47	1.32	1.31	2.09
EP 3500 average training time [sec], 4 processors	31	47	68	69	136
Speedup	1.23	1.53	1.16	1.52	2.15

[11]SYNAPSE: Synthesis of neural algorithms on a parallel systolic engine.

processor; see Ref. 18 and Refs. 32–34. The peak performance of the SYNAPSE 3 is 1.28 billion multiplications plus 1.28 billion additions plus 40 million look-up-table calls of the hyperbolic tangent or inverse hyperbolic tangent function per second. The SYNAPSE 3 can be used as PC-coprocessor board without significant performance loss. In this mode, the SYNAPSE 3 onboard memory (16 + 16 MBytes) is used to buffer (sub) matrices transferred from/to the PC memory. Up to three parallel SYNAPSE 3 boards can be used efficiently in one PC; see Ref. 34.

Robust optimal real-time guidance schemes $u^*_{\mathrm{app,I,I}}(z) - u^*_{\mathrm{app,XX}}(z)$ for the shuttle reentry are synthesized with the best successfully trained perceptrons; see Eqs. (27)–(32). Behavior and stability of these schemes have to be examined in many simulations under various atmospheric conditions. Simulations are carried out by computing $u^*_{\mathrm{app,...}}(z)$ and integrating numerically the dynamic equations (9)–(14) using the initial conditions (15). On the one hand, this can be done offline by choosing air density variations in advance; see Fig. 6 for ten instances of air density fluctuations $\sigma(t)\rho(h(t))$, with $\sigma_{\min}(h(t)) \le \sigma(t) \le \sigma_{\max}(h(t))$. On the other hand a graphical simulation environment, developed by A. Heim (formerly at TU München) and the author, visualizes online the shuttle reentry during the simulation; see Fig. 7 and Ref. 12.

The atmospheric conditions can be influenced interactively by forcing the real-time guidance scheme to react immediately. The simulation environment pops up two windows. The first window serves as a graphical user

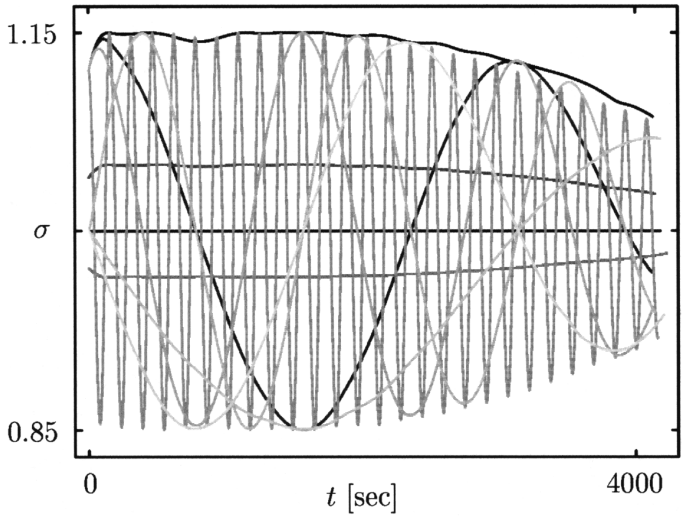


Fig. 6. Air density fluctuations.

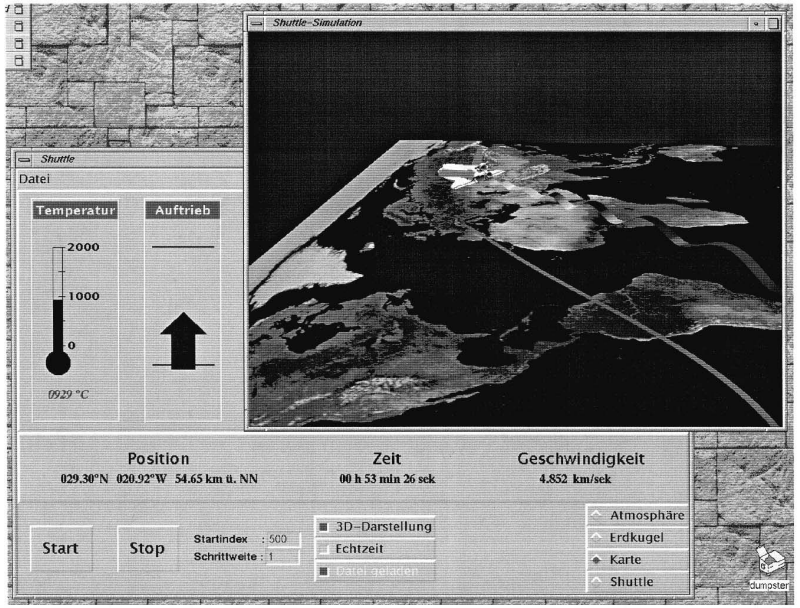


Fig. 7. Space shuttle reentry simulator.

interface and displays important data for the reentry maneuver. The second window shows a three-dimensional visualization of the shuttle during the reentry maneuver. The graphical user interface offers input and output functionality. Measurement instruments display all the state variables (e.g., the position of the shuttle) and the time steps of the simulation. The user can influence the air density with a slider in the upper right of the window setting the deviation from the nominal air density. In the upper left of the window, a thermometer displays the heating of the shuttle. Next to the right, the lift coefficient C_l is represented by an arrow. The length of the arrow indicates the value of C_l with $C_{l,\max}(z)$ marked by a horizontal line above the arrow. Next to the right, an artificial horizon gives information about the flight path angle γ and bank angle μ of the shuttle (covered in Fig. 7). The three-dimensional visualization window can be configured to show different views of the shuttle. The simulation of a complete reentry maneuver takes 3–5 minutes on a single 400 MHz Intel Pentium II processor with 100 MHz bus running on LINUX. An average of more than two frames per second for a realistic view of the flight can be achieved. Many simulations with different air density fluctuations $\sigma(t)\rho(h(t))$ [see Figs. 6 and

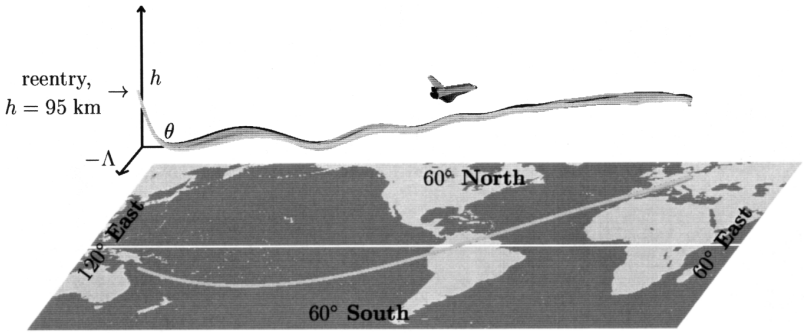


Fig. 8. Reentry trajectories using $u^*_{\text{app,I,I}}(z(t))$.

8 for ten instances of simulations] show that, for maximum reliability, a guidance scheme based on perceptrons with only one hidden neuron must be preferred.

These perceptrons have slightly lower accuracy compared to the best successfully trained perceptrons (see Table 1), but are less oscillatory by far (see Fig. 8). An oscillating $u^*_{\text{app,...}}(z)$ leads to a disastrous reentry maneuver outside the reentry tube for some $\sigma(t)\rho(h(t))$. In summary, the simulations show both the real-time capability and excellent robustness of the guidance scheme $u^*_{\text{app,I,I}}(z)$. All the constraints can be fulfilled and the final conditions (17) can be met approximately within the prescribed accuracy [i.e., with $\Phi(t_f) \lessapprox 1$]; see Eqs. (16) and (17).

4. Conclusions

Today, a large class of nonlinear robust optimal control problems arising in real-life applications is solvable numerically. The theoretical basis is provided by the theory of dynamic games which has been initiated by Isaacs in the early fifties and generalized during the last two decades. An efficient solution is possible only with a subsequent use of sophisticated numerical methods: (i) a direct method to solve auxiliary neighboring optimal control problems; (ii) an indirect method to solve multipoint boundary-value problems for ordinary differential equations; (iii) a high-dimensional synthesis method to obtain the robust optimal feedback controls for the relevant parts of the state space. The example outlined of an onboard reentry guidance for a space shuttle proves the real-life applicability of both the theory and numerical methods. The many computer simulations have offered high hopes that future space shuttles will perform better, i.e., have guaranteed lower maximum heating, lower maximum dynamic pressure, higher payload, and higher accuracy in meeting the final quasi-steady glide conditions. The universal approach presented here is not limited to guidance problems in aeronautics or astronautics. Optimal control problems with modeling inaccuracies and/or unpredictable and unmeasurable influences in mechanics, robotics, chemical engineering, and economics (to list only a few fields) can be solved as well.

References

1. Windhorst, R., Ardema, M. D., and Bowles, J. V., *Minimum Heating Entry Trajectories for Reusable Launch Vehicles*, Journal of Spacecraft and Rockets, Vol. 35, pp. 672–682, 1998.
2. Bradt, J., Langehough, M., and Robert, R., *Autonomous Guidance and Control for a Low L/D Crew Return Vehicle*, AIAA Paper 91-2819, 1991.
3. Chou, H. C., Ardema, M., and Bowles, J., *Near-Optimal Entry Trajectories for Reusable Launch Vehicles*, Journal of Guidance, Control, and Dynamics, Vol. 21, pp. 983–990, 1998.
4. Jänsch, C., and Markl, A., *Trajectory Optimization and Guidance for a Hermes-Type Reentry Vehicle*, AIAA Paper 91-2659, 1991.
5. Breitner, M. H., and Von Stryk, O., *Reentry Optimization of a European Space Shuttle*, Proceedings of the 8th Conference of the European Consortium for Mathematics in Industry (ECMI 94), Universität Kaiserslautern, Kaiserslautern, Germany, pp. 222–224, 1994.
6. Patankar, S. V., *Numerical Heat Transfer and Fluid Flow*, Hemisphere, Washington, DC, 1980.
7. Hartung, L. C., and Throckmorton, D. A., *Space Shuttle Reentry Heating Data Book: Volume 1, STS-2; Volume 2, STS-3; Volume 3, STS-5*; NASA Reference Publications 1191, 1192, 1193, NASA Langley Research Center, Hampton, Virginia, 1988.
8. Breitner, M. H., *Robust Optimal Onboard Reentry Guidance of a European Space Shuttle: Dynamic Game Approach and Guidance Synthesis with Neural Networks*, Deutsche Forschungsgemeinschaft, Schwerpunktprogramm "Echtzeit-Optimierung großer Systeme", Report 99-4, Clausthal-Zellerfeld, Germany, 1999; see http://www.zib.de/dfg-echtzeit/Publikationen/index.html.
9. Breitner, M. H., *Robust Optimal Feedback Controls: Dynamic Game Approach, Numerical Solution, and Real-Time Approximation*, Extended PhD Thesis, VDI-Verlag, Düsseldorf, Germany, 1996 (in German).
10. Breitner, M. H., *Construction of the Optimal Feedback Controller for Constrained Optimal Control Problems with Unknown Disturbances*, Computational Optimal Control, Edited by R. Bulirsch and D. Kraft, International Series of Numerical Mathematics, Birkhäuser, Basel, Switzerland, Vol. 115, pp. 147–162, 1994.

11. Breitner, M. H., *Real-Time Capable Approximation of Optimal Strategies in Complex Differential Games*, Proceedings of the 6th International Symposium on Dynamic Games and Applications, St. Jovite, Québec, Edited by M. Breton and G. Zaccour, Ecole des HEC, Montreal, Quebéc, Canada, 1994.
12. Breitner, M. H., and Heim, A., *Robust Optimal Control of a Reentering Space Shuttle*, Jahrbuch der Deutschen Gesellschaft für Luft- und Raumfahrt, Lilienthal-Oberth e.V., Bonn, Germany, Vol. 2, pp. 583–592, 1995.
13. Breitner, M. H., and Pesch, H. J., *Reentry Trajectory Optimization under Atmospheric Uncertainty as a Differential Game*, Annals of the International Society of Dynamic Games, Vol. 1, pp. 70–86, 1994.
14. Isaacs, R. P., *Differential Games, I: Introduction; II: Definition and Formulation; III: Basic Principles of the Solution Process; IV: Mainly Examples;* Research Memoranda RM-1391, RM-1399, RM-1411, RM-1486, The RAND Corporation, Santa Monica, California, 1954–55.
15. Isaacs, R. P., *Differential Games: A Mathematical Theory with Applications to Warfare and Pursuit*, *Control*, *and Optimization*, 4th Edition, Dover Publications, New York, NY, 1999.
16. Blaquière, A., Gerard, F., and Leitmann, G., *Quantitative and Qualitative Games*, Academic Press, New York, NY, 1969.
17. Başar, T., and Olsder, G. J., *Dynamic Noncooperative Game Theory*, 2nd Edition, Academic Press, London, England, 1995.
18. Breitner, M. H., *Synthesis of Robust Optimal Controls with SYNAPSE Neurocomputers*, Zeitschrift für Angewandte Mathematik und Mechanik, Vol. 79, Supplement 2, pp. S337–S338, 1999.
19. Ascher, U. M., Mattheij, R. M. M., and Russell, R. D., *Numerical Solution of Boundary-Value Problems for Ordinary Differential Equations*, Unabridged, Corrected Reprinting, SIAM Classics in Applied Mathematics, Philadelphia, Pennsylvania, Vol. 13, 1995.
20. Aziz, A. K., Editor, *Numerical Solutions of Boundary-Value Problems for Ordinary Differential Equations*, Academic Press, New York, NY, 1975.
21. Stoer, J., and Bulirsch, R., *Introduction to Numerical Analysis*, Springer, New York, NY, 1993.
22. Siemens AG, Breitner, M. H., and Pesch, H. J., *A Collision Avoidance Method for a Car against Another Car Based on Neural Networks*, German Patent No. 19534942, Deutsches Patentamt, München, Germany, 1998.
23. Von Stryk, O., *Numerical Optimal Control: Discretization*, *Nonlinear Optimization*, *and Computation of Adjoint Variables*, PhD Thesis, VDI-Verlag, Düsseldorf, Germany, 1994 (in German).
24. Von Stryk, O., *Numerical Solution of Optimal Control Problems by Direct Collocation*, Optimal Control, Calculus of Variations, Optimal Control Theory, and Numerical Methods, Edited by R. Bulirsch, A. Miele, J. Stoer, and K. H. Well, International Series of Numerical Mathematics, Birkhäuser, Basel, Switzerland, Vol. 111, pp. 129–143, 1993.
25. Gill, P. E., Murray, W., and Wright, M. H., *Practical Optimization*, 12th Printing, Academic Press, London, England, 1999.

26. Fletcher, R., *Practical Methods of Optimization*, 2nd Edition, Reprinting, John Wiley and Sons, New York, NY, 1999.
27. Deuflhard, P., and Apostolescu, V., *An Underrelaxed Gauss–Newton Method for Equality Constrained Nonlinear Least Squares Problems*, Optimization Techniques, Part 2, Edited by J. Stoer, Lecture Notes in Control and Information Sciences, Springer, Berlin, Germany, Vol. 7, pp. 22–32, 1978.
28. Hohmann, A., *Inexact Gauss–Newton Methods for Parameter-Dependent Nonlinear Problems*, PhD Thesis, Shaker, Aachen, Germany, 1994.
29. Nowak, U., and Weimann, L., *A Family of Newton Codes for Systems of Highly Nonlinear Equations: Algorithm*, *Implementation*, *Application*, Technical Report TR 91-10, Konrad-Zuse-Zentrum für Informationstechnik Berlin, Berlin, Germany, 1991.
30. Griewank, A., *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation,* SIAM, Philadelphia, Pennsylvania, 2000.
31. Dongarra, J. J., *Numerical Linear Algebra on High-Performance Computers* (*Software*, *Environments*, *Tools*), SIAM, Philadelphia, Pennsylvania, 1998.
32. Breitner, M. H., *Heuristic Option Pricing with Neural Networks and the Neurocomputer SYNAPSE 3*, Optimization, Vol. 47, pp. 319–333, 2000.
33. Breitner, M. H., and Bartelsen, S., *Training of Large Three-Layer Perceptrons with the Neurocomputer SYNAPSE*, Operations Research '98, Edited by P. Kall and H. J. Lüthi, Springer, Berlin, Germany, pp. 562–570, 1999.
34. Media Interface, *SYNAPSE 3 User Manual and Installation Guide; SynUse Base 3.1 Programming Manual*, *Reference Sheets*, *and Performance Report*, Media Interface GmbH, Dresden, Germany, 1998.

Optimization, 2000, Vol. 47, pp. 319–333
Reprints available directly from the publisher
Photocopying permitted by license only

© 2000 OPA (Overseas Publishers Association) N.V.
Published by license under
the Gordon and Breach Science
Publishers imprint.
Printed in Malaysia.

HEURISTIC OPTION PRICING WITH NEURAL NETWORKS AND THE NEURO-COMPUTER SYNAPSE 3

MICHAEL H. BREITNER*

TU Clausthal, Fachbereich Mathematik und Informatik, Erzstraße 1, D-38678 Clausthal-Zellerfeld, Germany

(Received 7 January 1999; In final form 22 January 1999)

Today's option and warrant pricing is based on models developed by Black, Scholes and Merton in 1973 and Cox, Ross and Rubinstein in 1979. The price movement of the underlying asset is modeled by continuous-time or discrete-time stochastic processes. Unfortunately these models are based on severely unrealistic assumptions. Permanently an unsatisfactory and quite artificial adaption to the true market conditions is necessary (future volatility of the underlying price). Here, an alternative heuristic approach with a highly accurate neural network approximation is presented. Market prices of options and warrants and the values of the influence variables form the usually very large output/input data set. Thousands of multi-layer perceptrons with various topologies and with different weight initializations are trained with a fast sequential quadratic programming (SQP) method. The best networks are combined to an expert council network to synthesize market prices accurately. All options and warrants can be compared to single out overpriced and underpriced ones for each trading day. For each option and warrant overpriced and underpriced trading days can be used to ascertain a better buy and sell timing. Furthermore the neural model gains deep insight into the market price sensitivities (option Greeks), *e.g.*, Δ, Γ, Θ and Ω. As an illustrative example we investigate BASF stock call warrants. Time series from the beginning of 1996 to mid 1997 of 74 BASF call warrant prices at the Frankfurter Wertpapierbörse (Frankfurt Stock Exchange) form the data basis. Finally a possible speed up of the training with the neuro-computer SYNAPSE 3 is briefly discussed.

Keywords: Neural networks; financial modeling; SQP-training methods; expert council networks; true market option and warrant pricing; inexpensive neuro-computers

Mathematics Subject Classifications 1991: Primary: 65Dxx, 68Txx, 90-XX; Secondary: 49M37, 90C30, 90A09, 68T05

*Phone/Fax: Germany + 5323 72 2959/+ 5323 72 2304, e-mail: breitner@math.tu ; clausthal.de

INTRODUCTION

Risk management is essential in a modern market economy. Financial markets enable firms and households to select an appropriate level of risk in their transactions by redistributing risks towards other agents who are willing and able to assume them. Markets for options, futures and other so-called derivative instruments – derivatives, for short – have a particular status. Futures allow agents to hedge against upcoming risks. These contracts promise future delivery of a certain item at a certain strike price. Options allow agents to hedge against one-sided risks. Options give the right, but not the obligation, to buy (call option) or sell (put option) something at a prespecified strike price at expiration (European style option) or any time up to expiration (American style option).

In October 1997 the Swedish Nobel Prize in Economic Sciences has been awarded to Myron S. Scholes, Stanford University, and Robert C. Merton, Harvard University. Inspired by the options and futures trading (Chicago Board Options Exchange) Scholes, Merton and Fischer Black, who died in 1995, study European stock options' pricing since the early seventies, see [3–5, 29–31] and [32]. The analytic pricing model is based on a continuous-time diffusion process (Ito process) for non-payout underlyings. In the late seventies Cox, Ross and Rubinstein used a numerical model instead to price also the more common American options, see [13, 14] and [15]. The diffusion process of the underlying price is simulated with a discretized time binomial tree (discrete-time stochastic process). Common to both models is that the correct option price is investigated by an option-underlying portfolio with risk-free profit. The theoretically fair option price p_{BS} (Black/Scholes) depends on

- the underlying price s,
- the strike price b,
- the time to expiration r,
- the risk-free interest rate ir up to expiration and
- the future volatility σ_s of the underlying price measured by its annualized standard deviation of percentage change in daily price.

Analogously p_{CRR} (Cox/Ross/Rubinstein) depends on s, b, r, ir and a discretized future volatility σ_s, *i.e.*, the probability of rising and falling s in each time step. During the last two decades both models have been improved in many ways. *E.g.*, American options and put options can be valued too and dividend yields of the underlying and variable interest rates can be handled, see [6, 16, 17, 38] and [40].

Today options on various underlyings are worldwide traded around-the-clock. Computer systems and networks that worldwide link offices and markets permit immediate exchange of true market information (securities exchanges and over-the-counter traders). The *theoretical pricing models*, *i.e.*, the Black/Scholes model and the Cox/Ross/Rubinstein model, are based on severely unrealistic assumptions:

(1) Markets are efficient, *i.e.*, nobody can consistently predict the direction of the market or an individual underlying. Underlying prices follow a memoryless continuous-time or discrete-time stochastic process;
(2) The future volatility σ_s of the underlying price can be estimated accurately and is a priori known to seller and buyer of an option.

Note that σ_s is the input which must be estimated for both models. A so-called implied volatility $\sigma_{s,i}$ can be fit artificially to equate model price and true market price obtained from the exchange or from over-the-counter traders. As a result, $\sigma_{s,i}$ often oscillates erratically and neither the Black/Scholes model nor the Cox/Ross/Rubinstein model captures option market conditions. In particular the very important option price sensitivities (option Greeks)

$$\Delta_* := \frac{\partial}{\partial s} p_*, \quad \Gamma_* := \frac{\partial^2}{\partial s^2} p_*, \quad \Theta_* := \frac{\partial}{\partial r} p_* \text{ and } \Omega_* := \frac{s}{p_*} \Delta_* \qquad (1)$$

usually are very inaccurate.[1] The estimation of options' risk and chance and a hedging for their issuers suffer from these inaccuracies.

HEURISTIC OPTION PRICING BASED ON NEURAL NETWORKS

In contrast to the theoretical pricing models a *heuristic pricing model* based on highly accurate neural network approximations can *learn*

[1] W. l. o. g. we assume a cover ratio of 1 for all options, *i.e.*, one option relates to one underlying.

true market pricing of options and warrants (options confirmed by a security), see [1, 7, 8, 10, 12, 27, 39] and [43] for theory and applications of neural networks. Like the theoretical option price $p_{BS}(s, b, r, ir, \sigma_s)$ or $p_{CRR}(s, b, r, ir, \sigma_s)$ the heuristic option price $p_h(s, b, r, t)$ depends on the permanently available underlying price s, strike price b[2] and time to expiration r. But instead of ir and the artificially estimated σ_s we use the permanently available trading day t as direct input for the heuristic pricing model. Note that $t \in \mathbb{R}^+$ enables a continuous-time model and an intra-day option pricing. Inputs and output of the neural network must be carefully transformed and equilibrated to facilitate the neural network training. The author's research showed[3] that input s should be replaced by moneyness s/b and that output p_h should be replaced by an option's premium $(b + p_h(t) - s(t))/s(t)$. With a consecutive numbering of the trading days, with the index set $I_{op} := \{1, 2, \ldots, n_{op}\}$ for the options considered, with the index sets I_i, $i \in I_{op}$, of traded days for option i and with the option market price $p_i(t)$, $i \in I_{op}$, $t \in I_i$, we define

$$
\begin{aligned}
b_{\min} &= \min_{i \in I_{op}} b_i, & b_{\max} &= \max_{i \in I_{op}} b_i,\\
t_{\min,i} &= \min I_i \quad \text{for } i \in I_{op}, & t_{\max,i} &= \max I_i \quad \text{for } i \in I_{op},\\
r_{\min} &= \min_{i \in I_{op}} r_i(t_{\max,i}), & r_{\max} &= \max_{i \in I_{op}} r_i(t_{\min,i}),\\
t_{\min} &= \min_{i \in I_{op}} t_{\min,i}, & t_{\max} &= \max_{i \in I_{op}} t_{\max,i},\\
s_{\min} &= \min_{t \in [t_{\min}, t_{\max}]} s(t),^4 & s_{\max} &= \max_{t \in [t_{\min}, t_{\max}]} s(t),^4\\
pr_{\min} &= \min_{i \in I_{op}, t \in I_i} \frac{b_i + p_i(t) - s(t)}{s(t)}, & pr_{\max} &= \max_{i \in I_{op}, t \in I_i} \frac{b_i + p_i(t) - s(t)}{s(t)}.
\end{aligned}
\tag{2}
$$

For $i = 1, 2, \ldots, n_{op}$ and $t = t_{\min,i}, t_{\min,i} + 1, \ldots, t_{\max,i}$ we get $m = t - t_{\min,i} + 1 + \sum_{l=1}^{i-1} \operatorname{card}(I_l)$ with $\operatorname{card}(I_l)$ the number of elements

[2]For the Black/Scholes and Cox/Ross/Rubinstein model there holds $p_*(cs, cb, r, i, \sigma_s) = cp_*(s, b, r, i, \sigma_s)$ for all $c \in \mathbb{R}^+$. Thus only input s or b must be considered. Nevertheless option market observations showed significant deviations from the equation above and both inputs s and b seem to be important for alternative models.

[3]A comprehensive discussion is omitted here to focus on the main results. However, it should be mentioned that the author tried out many input and output transformations and equilibrations.

[4]W. l. o. g. we assume that the underlying price is available every trading day.

of I_l and generate the input/output patterns (x_m, y_m) with $m \in I_p$, $I_p := \{1, 2, \ldots, n_p\}$, $n_p := \sum_{l=1}^{n_{op}} \operatorname{card}(I_l)$,

$$x_{m1} = 2\frac{t - t_{\min}}{t_{\max} - t_{\min}} - 1, \tag{3}$$

$$x_{m2} = 1 + 2\frac{\ln(r_i(t)) - \ln(r_{\max})}{\ln(r_{\max}) - \ln(r_{\min})}, \tag{4}$$

$$x_{m3} = 1 + 2\frac{\ln\frac{s(t)}{b_i} - \ln\frac{s_{\max}}{b_{\min}}}{\ln\frac{s_{\max}}{b_{\min}} - \ln\frac{s_{\min}}{b_{\max}}}, \tag{5}$$

$$x_{m4} = 1 + 2\frac{\ln(b_i) - \ln(b_{\max})}{\ln(b_{\max}) - \ln(b_{\min})}, \tag{6}$$

$$y_m = 1.9\frac{\frac{b_i + p_i(t) - s(t)}{s(t)} - pr_{\min}}{pr_{\max} - pr_{\min}} - 0.95. \tag{7}$$

For all $m \in I_p$ there holds $x_m \in [-1, 1]$[4] for the input vectors and $y_m \in [-0.95, 0.95]$ for the target. Note that the logarithm in the equilibration (4)–(6) ensures a *relative accuracy* instead of an absolute accuracy. *E.g.*, a 20% longer time to expiration means that $r_I := 50$ trading days increases to $r_I^- = 60$ trading days with $r_I^- - r_I = 10$ trading days and $r_{II} = 400$ trading days increases to $r_{II}^- = 480$ trading days with $r_{II}^- - r_{II} = 80$ trading days. But the input x_{m2}, see Eq. (4), increases by the same amount in both cases.

Here topological appropriate neural networks are given, *e.g.*, by three-layer perceptrons with or without shortcuts, with an auxiliary neuron N_0 (bias neuron) with $x_{m,0} := 1$, with input neurons $N_1 - N_4$, with hidden neurons $N_5 - N_{4+n_h}$, $1 \le n_h \le 5$, with output neuron N_{5+n_h}[5] and with hyperbolic tangent transfer functions

$$on_{m,i} := x_{m,i} \quad \text{for } i = 0, 1, \ldots, 4 \quad \text{and} \quad \text{for all } m \in I_p, \tag{8}$$

[5]Only one output neuron is advisable to facilitate the neural network training (separate neural networks in the case of more than one output variable).

$$on_{m,i} := \tanh\left(\sum_{j=0}^{4} w_{j,i}\, on_{m,j}\right) \quad \text{for } i = 5,\ldots,4+n_h \quad \text{and} \tag{9}$$

$$on_{m,5+n_h} := \tanh\left(\sum_{j=0}^{4+n_h} w_{j,5+n_h}\, on_{m,j}\right). \tag{10}$$

Trainable weights $w_{j,i}$ of the weight matrix $W \in \mathbb{R}^{5+n_h} \times \mathbb{R}^{n_h+1}$ are used. For perceptrons without shortcuts all weights $w_{j,5+n_h}$, $j \in \{1,2,3,4\}$, are frozen to $w_{j,i} \equiv 0$. Pattern set $P := \{(x_1, y_1), (x_2, y_2), \ldots, (x_{n_p}, y_{n_p})\}$ is divided into training set P_t and cross-validation set P_v with index sets $I_t := \{1, 2, \ldots 50, 101, 102, \ldots 150, 201, 202, \ldots\}$ and $I_v := \{51, 52, \ldots 100, 151, 152, \ldots 200, 251, 252, \ldots\}$. The approximation quality of the neural network can be estimated with the error functions

$$\varepsilon_t(W) := \frac{1}{2}\sum_{m \in I_t} (on_{m,5+n_h} - y_m)^2, \tag{11}$$

$$\varepsilon_v(W) := \frac{\text{card}(I_t)}{\text{card}(I_v)} \frac{1}{2} \sum_{m \in I_v} (on_{m,5+n_h} - y_m)^2. \tag{12}$$

As usual the perceptron is trained iteratively, *i.e.*, ε_t is decreased by adaption of W, as long as $\varepsilon_v < \varepsilon_t$ or $\varepsilon_v \approx \varepsilon_t$ holds (prevention of overtraining). Weight upgrades $W_{\text{new}} - W$ can be calculated with any minimization algorithm, *e.g.*, with usually favored first derivative methods such as the steepest descent or with second derivative methods such as Newton method. For first derivative methods generally we have the iterative sequence

$$W_{\text{new}} = W + \eta(\varepsilon_t(W), \text{grad}_W\, \varepsilon_t(W))\, \Delta W(\varepsilon_t(W), \text{grad}_W\, \varepsilon_t(W)) \tag{13}$$

with search direction ΔW and with step length η. Methods for constrained nonlinear least-squares problems, compare (11), are sequential quadratic programming (SQP) methods and generalized Gauß-Newton (GGN) methods which can exploit the special structure of the Hessian matrix of ε_t see [18, 19, 21–23, 26] and [34]. SQP and GGN methods usually approximate the Hessian matrix of ε_t by finite-differences

and update formulas, *i.e.*, become first derivative methods, and can deal with box constraints on the weights, linear constraints and smooth nonlinear constraints. Advantages of these methods are:

- A much better search direction ΔW is calculated in comparison to common training methods, *e.g.*, $\Delta W := -\text{grad}_W \varepsilon_t$ for the gradient method (backpropagation);
- The step length η is optimized permanently in comparison to common training methods with fixed step length. Thus the number of learning steps is reduced significantly (factor 10–1000000);
- Only ε_t and $\text{grad}_W\, \varepsilon_t$ are required which mainly can be computed by very fast matrix operations. For special topologies $\text{grad}_W\, \varepsilon_t$ also can be deduced easily by automatic differentiation, see [25, 33] and [36];
- Maximum and minimum of each weight can be set easily (box constraints);
- The total curvature of the neural network can be constrained (prevention from neural network oscillations);
- Convexity and monotonicity constraints can be set easily.

Note that usually the training stops before a local minimum of ε_t is reached due to an increasing ε_v with $\varepsilon_v > \varepsilon_t$. SQP and GGN methods automatically can overcome most of the training problems of three-layer perceptrons, *e.g.*, flat spots or steep canyons of the error function ε_t. Nevertheless SQP and GGN methods also only converge to one of the *very many local minima* of ε_t, see [2, 24] and [42]. For large training sets P_t often thousands, ten thousands or more local minima exist even for three-layer perceptrons with only few input and hidden neurons. Therefore the more neural networks with different weight initializations are trained, the more likely it is to train a perceptron successfully, *i.e.*, to find a good local minimum.

EXAMPLE: BASF CALL WARRANTS

We take true market prices (Frankfurter Wertpapierbörse) for BASF stocks and 74 BASF stock call warrants, compare Table I, for the period January 1, 1996, to July 4, 1997. With the help of a historical database and with Table I it is possible to restore the output/input data set.

TABLE I BASF call warrants investigated

Issuer	*German securities code number (WKN)*
BASF	870732
SBC Warburg	783606-08, 787995-97, 789618-20, 789647-49
Schweizerische Bankgesellschaft	789229-30, 789379, 789380, 789395-96, 790806-08
Trinkhaus and Burkhardt	814896, 817046-47, 817152-53, 817206-09
Citibank	815461, 818114-16, 818367-70
Rabobank	800200, 800202, 800404, 800406, 816410, 816412 816414
Commerzbank	816219-21, 816265-66, 816367
Lehman Brothers	900644, 902971, 904075, 904077, 904079-80
Deutsche Morgan Grenfell	556812, 557500, 560062, 560064
Société Générale Société Générale	726840-41, 726878-79
Salomon Brothers	718364, 729443, 729496
Merrill Lynch	801668-69
Sal. Oppenheim jr. and Cie	819684-85
Goldman Sachs	592049
Total number	74

There holds $t_{\min/\max}$ = 1st/376 trading day, $r_{\min/\max}$ = 49/1321 trading days, $b_{\min/\max}$ = 30.80/80.00 DM, $s_{\min/\max}$ = 32.56/69.10 DM and $pr_{\min/\max} = -0.0623/0.3903$. The 6093 training patterns and 4284 validation patterns, compare (3)–(7), form the large output/input data sets P_t and P_v, respectively. Three-layer perceptrons with and without shortcuts and 1 to 5 hidden neurons have appropriate topology to enable a highly accurate neural network approximation. With a SQP method, see [21, 23] and [22], 1000 different weight initializations are trained for each network topology. An average training time of 4 to 20 seconds on an IBM RS6000-43P/240 workstation is required for each initialization, compare Table II.

Note that the training for only few weight initializations leads to good neural networks, *i.e.*, into the neighborhood of a good local minimum without overtraining. For the best network with simple topology I we have $\varepsilon_t^* = 18.84187873$. Out of the 1000 trained networks 30 networks, *i.e.*, 3%, end up with $\varepsilon_t \in [18.84187873, 18.86072061]$, 126 networks end up with $\varepsilon_t \in [18.84187873, 20.72606660]$ and 836 networks end up with $\varepsilon_t \geq 37.68375746$ (very unsatisfactory convergence), compare the last block of Table II. For the best network with topology X we have $\varepsilon_t^* = 13.19227438$. Out of the 1000 trained networks only 4 networks end up with $\varepsilon_t \in [13.19227438, 14.51150182]$, 10 networks end up with $\varepsilon_t \in [13.19227438, 19.78841157]$ and 975 networks end up with $\varepsilon_t \geq 26.38454876$ (very unsatisfactory convergence).

TABLE II Offline SQP-training of neural networks with different topology

Topology	*I*	*II*	*III*	*IV*	*V*	*VI*	*VII*	*VIII*	*IX*	*X*
n_h	1	1	2	2	3	3	4	4	5	5
Shortcuts	no	yes	no	yes	no	yes	no	yes	no	yes
B. n. n.[1]	nn_I^*	nn_{II}^*	nn_{III}^*	nn_{IV}^*	nn_V^*	nn_{VI}^*	nn_{VII}^*	nn_{VIII}^*	nn_{IX}^*	nn_X^*
N. t. W.[2]	7	11	13	17	19	23	25	29	31	35
A. t. t.[3] in seconds	4	5	9	11	12	14	16	17	20	20
ε_t^*[4]	18.84	15.84	14.86	15.70	14.78	13.75	14.30	14.24	13.55	13.19
$\frac{\varepsilon_v - \varepsilon_t^*}{\varepsilon_t^*} < 0.001$ in %[4]	18.70	14.27	14.62	13.50	14.24	13.34	14.21	13.31	12.98	11.53
$\frac{\varepsilon_v - \varepsilon_t^*}{\varepsilon_t^*} < 0.1$ in %	3.0	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
$\frac{\varepsilon_v - \varepsilon_t^*}{\varepsilon_t^*} < 0.2$ in %	12.6	0.3	0.6	0.9	0.7	0.2	0.6	0.5	0.3	0.4
$\frac{\varepsilon_v - \varepsilon_t^*}{\varepsilon_t^*} < 0.3$ in %	14.2	0.5	1.2	1.2	1.2	0.5	1.0	0.9	0.6	0.4
$\frac{\varepsilon_v - \varepsilon_t^*}{\varepsilon_t^*} < 0.5$ in %	15.2	0.7	4.0	1.6	3.2	1.3	1.7	1.5	1.2	0.7
$\frac{\varepsilon_v - \varepsilon_t^*}{\varepsilon_t^*} < 1.0$ in %	16	0.8	12.2	2.2	12.4	2.3	9.9	2.6	7.5	1.0
$\frac{\varepsilon_v - \varepsilon_t^*}{\varepsilon_t^*} < 9.0$ in %	16.4	2.4	15.4	4.3	15.4	4.0	12.6	5.3	10.9	2.5
$\frac{\varepsilon_v - \varepsilon_t^*}{\varepsilon_t^*} < 99.0$ in %	42.7	74.4	36.1	64.8	36.2	55.2	32.4	50.2	30.6	41.3
$\frac{\varepsilon_v - \varepsilon_t^*}{\varepsilon_t^*} < 99.0$ in %	93.3	76.8	83.9	68.2	80.6	60.6	71.2	57.4	68.6	49.0

[1] Best neural network;
[2] Number of trainable weights;
[3] Average training time per neural network on an IBM RS6000-43P/240 Mhz work-station;
[4] minimum error on P_t and P_v, respectively, for nn_-^*.

To reduce erratic oscillations and to improve accuracy the best networks with different topology are combined to an expert council network

$$nn^*(s,b,r,t) = \frac{1}{10}\sum_{i=1}^{X} nn_i^*(s,b,r,t) \qquad (14)$$

to synthesize market prices of options and of warrants, compare Figure 1.

Consider BASF calls on July 4, 1997, and a BASF stock price $s = 66.18$ DM. For an out of the money strike $b = 70$ DM a BASF call market price is 10 DM for 636 trading days (2 years and 6.4 months) to expiration, is 5 DM for 164 trading days to expiration, is 2.50 DM for 80 trading days to expiration and is 1.25 DM for 57 trading days to expiration, compare Figure 1 again.

For 251 trading days (1 year) to expiration a BASF call market price is 5 DM for a strike $b = 76.80$ DM, is 10 DM for a strike $b = 62$ DM, is 15 DM for a strike $b = 53.70$ DM, is 20 DM for a strike $b = 47.50$ DM, is 25 DM for a strike $b = 42.10$ DM, is 30 DM for a strike $b = 36.70$ DM and is 35 DM for a strike $b = 31.10$ DM. With $nn^*(s,b,r,t)$

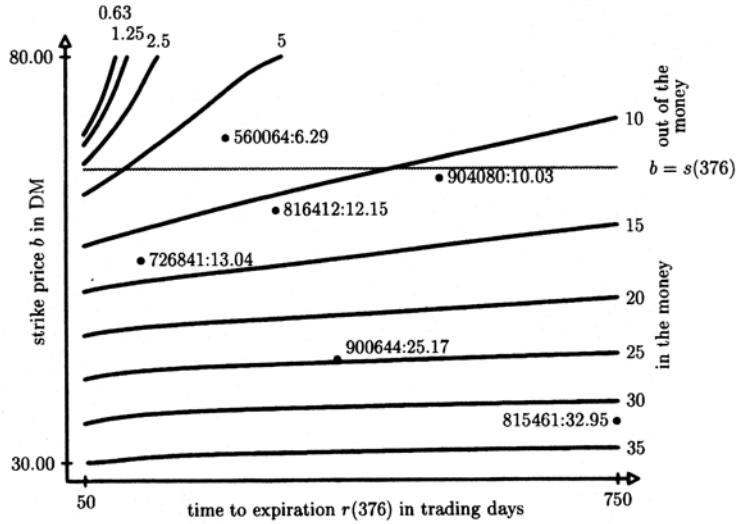


FIGURE 1 Contour plot for the heuristic BASF call warrant price in DM (right/top numbers) on July 4, 1997.

all options and warrants can be compared to single out overpriced and underpriced ones for each trading day. For each option and warrant overpriced and underpriced trading days can be used to ascertain a better buy and sell timing. Six warrants are marked in Figure 1 – •, WKN and market price – to rate the warrants' price. Note that on July 4, 1997, the price of warrant 904080 issued by Lehman Brothers is 10.03 DM and 0.96 DM, *i.e.*, 8.7%, below p_h.

Furthermore the neural model gains deep insight into the market price sensitivities, *e.g.*, Δ_h, Γ_h, Θ_h and Ω_h, compare Figure 2.

Again six warrants are marked – • and WKN – in Figure 2 to rate the warrants' Ω_h (gearing, leverage effect). On July 4, 1997, consider the BASF call 560064 issued by the Deutsche Morgan Grenfell (DMG). We have a strike $b = 70$ DM, a time to expiration $r = 237$ trading days and at noon a BASF stock price $s = 66.18$ DM. From the heuristic pricing model $nn^*(s,b,r,t)$ we obtain $p_h = nn^* = 6.74$ DM, $\Delta_h = 0.38$, $\Gamma = 0.016$/DM, $\Theta = -0.014$ DM/trading day and $\Omega_h = 3.8$, compare Definition (1) and Figure 2. The lower BASF call market price obtained most likely is due to the so-called spread, *i.e.*, the DMG sells/takes back the BASF call 560064 for 7.09 DM/6.29 DM (spread of

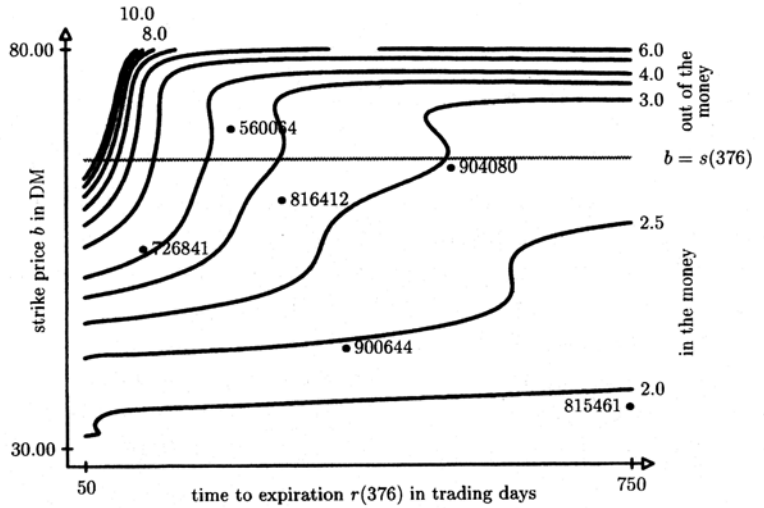


FIGURE 2 Contour plot for Ω_h of BASF call warrants (right/top numbers) on July 4, 1997.

0.80 DM or 12%). The issuer DMG usually wants to hedge the sold warrants, *i.e.*, wants to manage a risk free portfolio. Thus $\Delta_h = 0.38$ implies that for 100 warrants sold the DMG has to hold 38 BASF stocks. $\Omega_h = 3.8$ indicates that an increase/a decrease of the BASF stock price to 68/64 DM, *i.e.*, +2.75%/−3.29%, approximately results in an increase/a decrease of the BASF call price p_h to 7.44 DM/5.90 DM, *i.e.*, +10.45%/−12.52% (linearized leverage effect). More accurate than before $\Delta_h = 0.38$ *and* $\Gamma = 0.016$/DM indicate that the same change of the BASF stock price results in an increase/a decrease of the BASF call price p_h to 7.47 DM/5.94 DM (quadratic approximation of the leverage effect). $\Theta = -0.014$ DM/trading day says that p_h decreases by 0.014 DM every trading day due to the natural decrease of the time to expiration, if the other influence variables do not change. The good extrapolation properties of nn^* for up to 2 future weeks facilitate the estimation of options' risk and chance and the hedging of/with options.

The user friendly software package *Warrant-Pro* with graphical user interface, which will provide a framework for professional true market pricing of options and warrants, is under development in cooperation with the Market Maker Software, Kaiserslautern, and the Dresdner Bank, Frankfurt.

SPEED UP OF THE SQP-TRAINING WITH THE SYNAPSE 3

The neural networks for the above mentioned real-life application are computationally very expensive, compare Table II. The training of all neural networks $nn_1 - nn_X$ takes more than one day on todays high performance workstations and PCs. With $A_1 := (x_1 x_2 \dots x_{6093}) \in \mathbb{R}^{5,6093}$, $W_{12} \in \mathbb{R}^{n_h,5}$ (weights from the input layer to the hidden layer), $W_{23} \in \mathbb{R}^{1,n_h}$ (weights from the hidden layer to the output), $W_{13} \in \mathbb{R}^{1,5}$ (shortcuts) and $Y_s := (y_1 y_2 \dots y_{6093}) \in \mathbb{R}^{1,6093}$ the error function's gradient $(\partial/\partial W)\varepsilon_t$ is computable based on the matrix arithmetic algorithm

$$\begin{aligned} N_2 &:= W_{12}A_1,\ A_2 := \tanh(N_2), \\ N_3 &:= W_{23}A_2 + W_{13}A_1,\ A_3 := \tanh(N_3), \end{aligned} \tag{15}$$

$$S_3 := A_3 - Y_s,\ D_3 := \tanh'(N_3),\ E_3 := S_3 \odot D_3, \tag{16}$$

$$S_2 := W_{23}^T E_3,\ D_2 := \tanh'(N_2),\ E_2 := S_2 \odot D_2, \tag{17}$$

$$G_{12} := E_2 A_1^T,\ G_{13} := E_3 A_1^T,\ G_{23} := E_3 A_2^T, \tag{18}$$

with $(\partial/\partial W)\varepsilon_t$ in G_{12}, G_{23} and G_{13}. Note that $\odot$, tanh () and tanh′() denote the elementary matrix operations. For each $(\partial/\partial W)\varepsilon_t$ 158418 $n_h + 121860$ matrix multiplication operations, $12186\,n_h + 12186$ tanh () and tanh′() calls and $6093\,n_h + 18279$ elementary matrix operations must be computed. An acceptable computing time can be achieved on the one hand by very expensive parallel and vector high performance computers, see [20, 35, 41] and [44]. On the other hand specialized neuro-computers, which support parallel or matrix algorithms in hardware, can speed up compute power up to two orders of magnitude inexpensively, too. We focus on the newly developed SYNAPSE 3[6] neuro-computer PCI-board based on the patented MA16 neuro-signal processor, see [9–12, 28] and [37]. For multiplication (16 bit fixed-point) and addition (48 bit fixed-point) of large matrices the SYNAPSE3 is up to 20 times faster than todays high performance workstations and PCs. The peak performance is 2.56 billion multiplications or additions plus 40 million look-up-table calls of predefineable functions per second. All SYNAPSE 3 computations must be implemented with the C++ class library SynUse•Base.

Acknowledgments

The author gratefully appreciates the support by the Market Maker Software GmbH, Kaiserslautern (*Data-Pool* financial database), P. E. Gill, University of California San Diego (SQP methods), and the MediaInterface GmbH, Dresden (SYNAPSE 3 cooperation). Last but not least, the author thanks both reviewers for their valuable comments and constructive criticism which helped to improve the paper significantly.

[6] Synthesis of neural algorithms on a parallel systolic engine.

References

[1] Anderson, J. A. (1995). *An Introduction to Neural Networks*. MIT Press, Cambridge (Massachusetts).

[2] Auer, P., Hebster, M. and Warmuth, M. K. (1996). Exponentially many local minima for single neurons. In: Touretzky, D. S., Mozer, M. C. and Hasselmo, M. E., Eds., *Proceedings of the Neural Information Processing Systems 1995*, MIT Press, Cambridge (Massachusetts).

[3] Black, F. and Scholes, M. (1972). The Valuation of Option Contracts and a Test of Market Efficiency. *Journal of Finance*, **27**.

[4] Black, F. and Scholes, M. (1973). The Pricing of Options and Corporate Liabilities. *Journal of Political Economy*, **81**.

[5] Black, F. (1989). How we came Up with the Option Formula. *Journal of Portfolio Management*, **15**.

[6] Bookstaber, R. M. (1991). Option pricing and investment strategies. McGraw-Hill, London.

[7] Breitner, M. H. (1996). *Robust optimale Rückkopplungssteuerungen gegen unvorhersehbare Einflüsse: Differentialspielansatz, numerische Berechnung und Echtzeitapproximation. Ph.D Thesis 1995*, Fortschritt-Bericht 596, Reihe 8 "Meß-, Steuerungs- und Regelungstechnik", VDI-Verlag, Düsseldorf.

[8] Breitner, M. H. (1996). *Neuronale Netze im Licht moderner Hardware. Proceedings of the Workshop Neuronale Netze und Neuro-Fuzzy-Systeme*, TU Clausthal, Clausthal-Zellerfeld.

[9] Breitner, M. H., Synthesis of Robust Optimal Controls with SYNAPSE Neurocomputers. To appear in ZAMM 1998.

[10] Breitner, M. H. and Bartelsen, S., Optimal portfolio management through forecasting of stronger stock price changes with the neuro-computer SYNAPSE. *Proceedings of the Sixth Viennese Workshop on Optimal Control*, Dynamic Games, Nonlinear Dynamics and Adaptive Systems, Wien 1997. Preprint: Mathematik-Bericht 97/11, Institut für Mathematik, TU Clausthal, Clausthal-Zellerfeld 1997.

[11] Breitner, M. H. and Bartelsen, S., Training of Large Three-layer Perceptrons with the Neurocomputer SYNAPSE. To appear in Kall, P., Ed., *Operations Research '98*, Springer, Berlin 1999.

[12] Breitner, M. H., Rettig, U. and Von Stryk, O., Robust Control with Large Neural Networks Emulated on the Neuro-computer Board SYNAPSE2•PC. In: Sydow, A., Ed., Application in Modelling and Simulation, Wissenschaft and Technik, Berlin 1997.

[13] Cox, J. C. and Ross, S. A. (1976). The Valuation of Options for Alternative Stochastic Processes. *Journal of Financial Economics*, **3**.

[14] Cox, J. C., Ross, S. A. and Rubinstein, M. (1979). Option Pricing: A Simplified Approach. *Journal of Financial Economics*, **7**.

[15] Cox, J. C. and Rubinstein, M., Options Markets. Englewood Cliffs, New Jersey 1985.

[16] Chriss, N. A., Black-Scholes and Beyond. McGraw-Hill, London 1996.

[17] Daigler, R. T., Advanced options trading: the analysis and evaluation of trading strategies, hedging tactics and pricing models. Probus Publisher, Chicago 1994.

[18] Deuflhard, P. and Apostolescu, V., A Gauss-Newton method for nonlinear least squares problems with nonlinear equality constraints. TU München, Report TUM-MATH-7607, 1976.

[19] Deuflhard, P. and Apostolescu, V., An underrelaxed Gauss-Newton method for equality constrained nonlinear least squares problems. In: *Optimization techniques. Lecture Notes in Control and Information Sciences 7*, Springer, Berlin 1978.

[20] Eckmiller, R., Ed., Parallel processing in neural systems and computers. North-Holland, Amsterdam 1990.

[21] Fletcher, R., Practical methods of optimization. Wiley, New York 1987.

[22] Gill, P. E., Murray, W. and Wright, M. H., Practical optimization. Academic Press, London, 1981.

[23] Gill, P. E., Murray, W. and Saunders, M. A., Large-scale SQP methods and their Application in Trajectory Optimization. In: Bulirsch, R. and Draft, D., Eds., Computational Optimal Control. Birkhäuser, *International Series of Numerical Mathematics*, **115**, Basel 1994.

[24] Gori, M. and Tesi, A. (1992). On the problem of local minima in backpropagation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **14**.

[25] Griewank, A. and Corliss, G. F., Eds., Automatic Differentiation of algorithms: Theory, Implementation and Application. SIAM, Philadelphia 1991.

[26] Hohmann, A., Inexact Gauss-Newton methods for parameter dependent nonlinear problems. *Ph.D Thesis*, Shaker, Aachen 1994.

[27] Hornik, K. (1991). Approximation Capabilities of Multilayer Feedforward Networks. Neural Networks, **4**.

[28] MediaInterface Dresden GmbH SYNAPSE 3 (Technische Beschreibung/Installationsvorschrift, Programmierhandbuch SynUseBase 3.1, Referenzhandbuch API, Performance Report). MediaInterface, Dresden 1998.

[29] Merton, R. C. (1971). Optimum Consumption and Portfolio Rules in a Continuous Time Model. *Journal of Economic Theory*, **3**.

[30] Merton, R. C. (1973). Theory of Rational Option Pricing. *Bell Journal of Economics and Management Science*, **4**.

[31] Merton, R. C. (1973). An Intertemporal Capital Asset Pricing Model. *Econometrica*, **41**.

[32] Merton, R. C., Option Pricing When Underlying Stock Returns are Discontinuous. *Journal of Financial Economics*, **3**.

[33] Monagan, M. B. and Neuenschwander, W. M., GRADIENT-Algorithmic Differentiation in Maple. Report, ETH Zürich, Institut für Wissenschaftliches Rechnen, 1992.

[34] Nowak, U. and Weimann, L., A Family of Newton Codes for Systems of Highly Nonlinear Equations – Algorithm, Implementation, Application. Technical Report TR 91-10, Konrad-Zuse-Zentrum für Informationstechnik Berlin, 1991.

[35] Pitas, I., Ed., Parallel algorithms for digital image processing, computer vision and neural networks. Wiley, Chichester (England) 1993.

[36] Rall, L. B., Automatic Differentiation – Techniques and Applications. Springer Lecture Notes in Computer Science, **120**, 1981.

[37] Ramacher, U. (1992). SYNAPSE-A Neurocomputer That Synthesizes Neural Algorithms on a Parallel Systolic Engine. *Journal of Parallel and Distributed Computing*, **14**.

[38] Redhead, K. Financial Derivates – An Introduction to Futures, Forewards, Options and Swaps. Prentice Hall, Herfordshire 1997.

[39] Rojas, R., Neural Networks – A Systematic Introduction. Springer, Berlin 1996.

[40] Stoll, H. R. and Whaley, R. E., Futures and options: theory and applications. South-Western Publications, Cincinnati 1993.

[41] Takefuji, Y., Neural network parallel computing. Kluwer, Boston 1992.

[42] Watrous, R. L., A comparison between squared error and relative entropy metrics using several optimization algorithms. Complex Systems, **6**, 1992.

[43] Zell, A., Simulation Neuronaler Netze. Addison-Wesley, Bonn 1996.

[44] Zell, A., Mamier, G., Mache, N. and Vogt, M., Simulation Neuronaler Netze auf SIMD- und MIMD-Parallelrechnern. In: Spies, P. P., Ed., *Euro-ARCH '93, Europäischer Informatik-Kongreß Architektur von Rechensystemen, München, 1993*, Springer Berlin 1993.

Mittelfristige Zinsprognose basierend auf technischen Ansätzen mit parallel trainierten Perzeptrons durch FAUN 0.2-PVM

Diplomarbeit von
Patrick Mehmert

Juli 2000

Technische Universität Clausthal
Institut für Mathematik

Referenten:
Prof. Dr. Th. Hanschke
Dr. M. H. Breitner

1 Einleitung

Neuronale Netze[1] finden seit vielen Jahren Anwendung in den verschiedensten Bereichen der Ingenieur-, Natur- und Wirtschaftswissenschaften. Sie gehören zusammen mit anderen Methoden, wie z.B. den Genetischen Algorithmen und der Fuzzy-Logic, zu Verfahren, deren Vorbilder der Natur entstammen und die den weitläufigen Begriffen der künstlichen Intelligenz und des Softcomputing zugeordnet werden können. Zwar können Neuronale Netze entgegen mancher vergangener Euphorie keine „Wunder" vollbringen, dennoch haben sich zahlreiche sinnvolle Anwendungen der Klassifikation, Identifikation und Prognose gefunden, in denen sie sich etabliert und bewährt haben.

Mathematisch gesehen dienen Neuronale Netze als universelle Approximatoren, die komplizierte nichtlineare Zusammenhänge zwischen bekannten Ein- und Ausgabegrößen durch unterschiedliche Trainingsmethoden „lernen" können. Nach dem Training können die Neuronalen Netze einerseits zur Interpolation unbekannter Werte genutzt werden, andererseits aber auch zur Extrapolation außerhalb der betrachteten Menge. Sofern das zu bearbeitende Problem eine zeitliche Abhängigkeit aufweist, können Neuronale Netze somit zur Prognose zukünftig eintretender Werte eingesetzt werden.

Damit wird insbesondere ein Einsatz bei finanzwirtschaftlichen Problemstellungen interessant, da eine zuverlässige Prognose von Marktdaten, wie zum Beispiel Aktienkursen oder Zinssätzen, sinnvolle unternehmerische Entscheidungen erleichtern. Daraus ergeben sich ganz konkret die Möglichlichkeiten einer Gewinnsteigerung bzw. einer Verlustbegrenzung. Diese können aber nur dann ideal genutzt werden, wenn der Vorteil eines Informationsvorsprungs vor anderen Marktteilnehmern besteht. An dieser Stelle stellt sich die bisher nicht eindeutig gelöste Frage, inwieweit die Theorie effizienter Märkte Gültigkeit besitzt, welche besagt, daß die Kurse von Finanztiteln bereits alle verfügbaren Informationen reflektieren. In diesem Fall wäre es gar nicht möglich, systematisch Gewinne über der allgemeinen Marktentwicklung zu erzielen. Innerhalb dieser Arbeit kann unabhängig von der Frage, ob und in welchem Maße eine Informationseffizienz vorliegt, davon ausgegangen werden, daß ein innovatives, nicht allgemein zugängliches Prognoseverfahren in jedem Fall eine günstigere Position als Marktteilnehmer ermöglicht.

Die Untersuchungen dieser Arbeit gliedern sich in zwei Teile. Zunächst wird eine mit dem frei verfügbaren Softwarepaket PVM[2] parallelisierte Version des Programms FAUN 0.2[3] zum Training Neuronaler Netze vorgestellt. Da bei der hier verwendeten Trainingsmetho-

[1] Kurz NN, oft auch richtiger als *künstliche* neuronale Netze (KNN) bezeichnet.
[2] **P**arallel **V**irtual **M**achine, public domain Programmpaket, Version 3.4.
[3] **F**ast **A**pproximation with **U**niversal neural **N**etworks, seit 1997 von Dr. M. H. Breitner entwickelt. Alle Nennungen des Programms in dieser Arbeit beziehen sich auf die Version 0.2, die erweiterte Version 0.3 steht kurz vor der Fertigstellung.

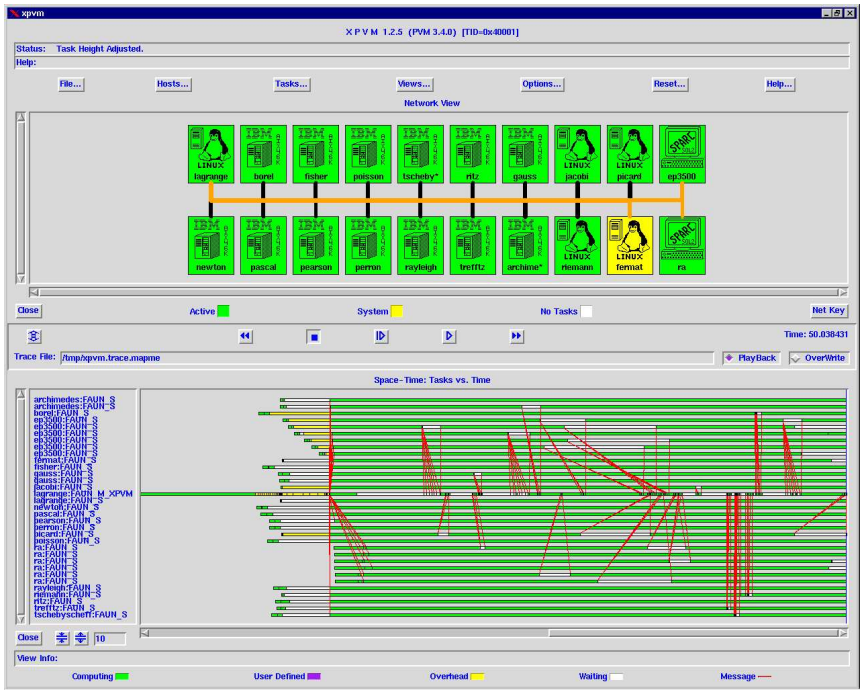


Abbildung 1: Visualisierung des parallelen Trainings Neuronaler Netze mit FAUN-PVM durch die grafische Schnittstelle XPVM. Oben ist die verwendete inhomogene Rechnerkonfiguration zu erkennen, im unteren Bereich die parallel arbeitenden Prozesse, bei der in der Mitte der Master dargestellt ist, welcher die administrativen Aufgaben übernimmt. Dieser kommuniziert mit den Slaves, welche dezentral die eigentliche Rechenarbeit leisten.

de ca. 100–10000 Netze initialisiert und trainiert werden, um mit hoher Wahrscheinlichkeit eine gute Approximationsfunktion zu erhalten, ist ein entsprechender Rechenaufwand nötig. Dieser liegt für eine Topologie, d.h. eine bestimmte Netzstruktur, bei heutige üblichen Rechnern (Pentium II PC mit 500 MHz) im Stunden- bis Tagesbereich und kann durch das parallele Training verschiedener Netze fast verlustfrei um den Faktor der zusätzlichen Rechenleistung reduziert werden. Um eine unabhängige Entwicklung von paralleler und skalarer Version von FAUN zu gewährleisten, wurde eine grobe Parallelisierung gewählt. Das eigentliche Trainingsprogramm FAUN bleibt unverändert und wird zur Unterroutine des PVM-Mantelprogramms FAUN_S - dem sogenannten „Slave". Dieser empfängt vom „Master" FAUN_M alle nötigen Informationen zum Netztraining und sendet die berechneten Ergebnisse zurück. Der Master verteilt solange Trainingsaufträge bis eine geforderte Mindestanzahl von trainierten Netzen eingetroffen ist. Da diese administrativen Aufgaben

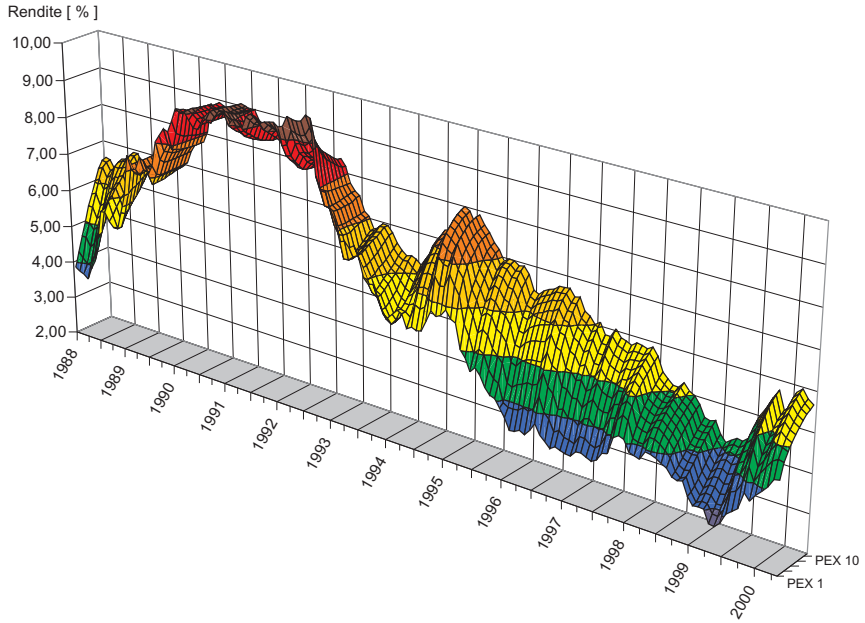
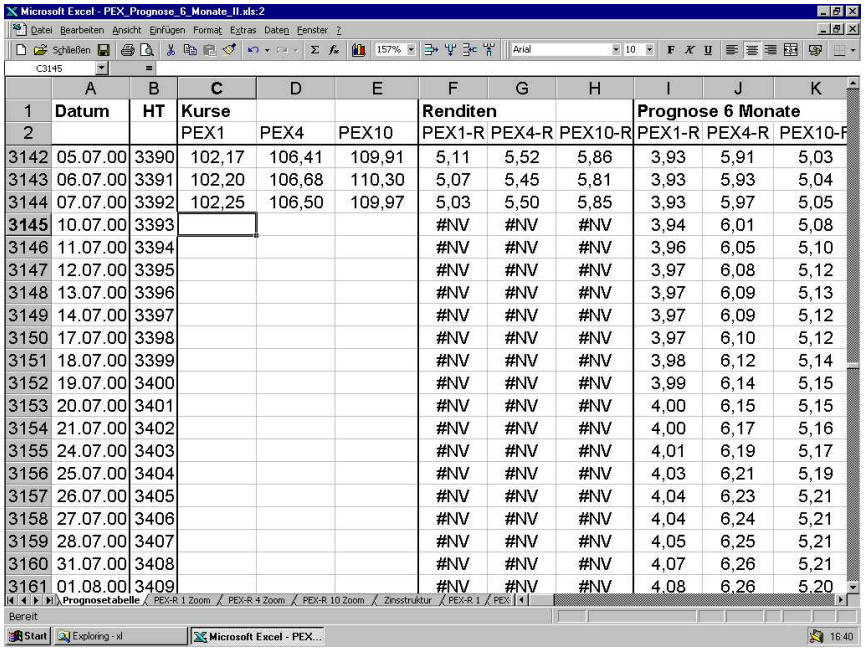


Abbildung 2: Renditen des Deutschen Pfandbriefindex PEX über Laufzeiten von einem bis zehn Jahren für den Zeitraum von 1988 bis Mitte April 2000, welche die Grundlage eines Zinsprognosemodells bilden sollen.

kaum Rechenleistung in Anspruch nehmen, können auch auf dem Rechner des Masters noch sinnvoll Slaves eingesetzt werden.

Im zweiten Teil der Arbeit wird dieses neue FAUN-PVM Programmpaket benutzt, um aus einer Reihe von Modellen zur Zinsprognose das vielversprechendste herauszufinden. Nur durch die starke Verkürzung der Rechenzeit konnten viele verschiedene Modelle in angemessener Zeit berechnet werden. Das endgültige Modell dient zur kurz- bis mittelfristigen Prognose der deutschen Pfandbrief-Renditen für drei, sechs und zwölf Monate, die sich aus den Kursen des Pfandbriefindex PEX ergeben. Um die Zinsstruktur über die Laufzeiten abbilden zu können, wurden die PEX-Renditen für Laufzeiten von ein, vier und zehn Jahren in die Prognose aufgenommen, während die Werte für die übrigen Laufzeiten quadratisch interpoliert werden. Dem Training der Netze lagen die historischen Kursreihen des PEX seit Anfang 1988 zugrunde, so daß die hier vorgestellte Prognose eine technische Analyse der Kursverläufe darstellt, im Unterschied zu einer Fundamentalanalyse, bei der weitere volks- und finanzwirtschaftliche Einflußfaktoren berücksichtigt werden. Das endgültige Modell wurde benutzerfreundlich in die weit verbreitete Tabel-



	A	B	C	D	E	F	G	H	I	J	K
1	**Datum**	**HT**	**Kurse**			**Renditen**			**Prognose 6 Monate**		
2			PEX1	PEX4	PEX10	PEX1-R	PEX4-R	PEX10-R	PEX1-R	PEX4-R	PEX10-R
3142	05.07.00	3390	102,17	106,41	109,91	5,11	5,52	5,86	3,93	5,91	5,03
3143	06.07.00	3391	102,20	106,68	110,30	5,07	5,45	5,81	3,93	5,93	5,04
3144	07.07.00	3392	102,25	106,50	109,97	5,03	5,50	5,85	3,93	5,97	5,05
3145	10.07.00	3393				#NV	#NV	#NV	3,94	6,01	5,08
3146	11.07.00	3394				#NV	#NV	#NV	3,96	6,05	5,10
3147	12.07.00	3395				#NV	#NV	#NV	3,97	6,08	5,12
3148	13.07.00	3396				#NV	#NV	#NV	3,97	6,09	5,13
3149	14.07.00	3397				#NV	#NV	#NV	3,97	6,09	5,12
3150	17.07.00	3398				#NV	#NV	#NV	3,97	6,10	5,12
3151	18.07.00	3399				#NV	#NV	#NV	3,98	6,12	5,14
3152	19.07.00	3400				#NV	#NV	#NV	3,99	6,14	5,15
3153	20.07.00	3401				#NV	#NV	#NV	4,00	6,15	5,15
3154	21.07.00	3402				#NV	#NV	#NV	4,00	6,17	5,16
3155	24.07.00	3403				#NV	#NV	#NV	4,01	6,19	5,17
3156	25.07.00	3404				#NV	#NV	#NV	4,03	6,21	5,19
3157	26.07.00	3405				#NV	#NV	#NV	4,04	6,23	5,21
3158	27.07.00	3406				#NV	#NV	#NV	4,04	6,24	5,21
3159	28.07.00	3407				#NV	#NV	#NV	4,05	6,25	5,21
3160	31.07.00	3408				#NV	#NV	#NV	4,07	6,26	5,21
3161	01.08.00	3409				#NV	#NV	#NV	4,08	6,26	5,20

Abbildung 3: Ausschnitt des Eingabeblattes der in EXCEL 97 implementierten Prognosemodelle: Die leicht zugänglichen Kursdaten werden automatisch in die Renditen umgerechnet und die Prognose entsprechend errechnet. Um eine bessere Anschauung zu erhalten und Entwicklungen besser beurteilen zu können, sind zahlreiche grafische Darstellungen der Prognosen wesentlicher Bestandteil der Implementation. Diese fanden auch Verwendung in dieser Arbeit, wie vor allem in 4.2.1, 4.2.2 und dem Anhang zu sehen ist.

lenkalkulation EXCEL 97 implementiert und der Kreissparkasse Clausthal-Zellerfeld zur Verfügung gestellt, die praktische Erfahrungen in der Anwendung des Systems und den erzielten Prognosegüten sammeln wird.

4 Zinsprognose mit Neuronalen Netzen

4.1 Grundlagen und Zielsetzungen

Da das Verhältnis von Einlagen und auszugebenden Krediten nicht immer ausgeglichen sein kann, müssen Kreditinstitute in gewissen Abständen selbst Geld am Kapitalmarkt aufnehmen, um sich zu refinanzieren. Die erwartete Entwicklung des Zinsniveaus ist dabei von entscheidender Bedeutung für die angewandte Refinanzierungsstrategie. Hierbei können grob zwei Fälle unterschieden werden:

- Erwartung eines Zinsanstiegs:
  Steigen die Zinsen, ist es günstig zum aktuellen Zeitpunkt höhere Geldvolumina zum niedrigeren Zins für längere Zeiträume aufzunehmen. Nicht benötigte Reserven müßten wieder angelegt werden, wobei zu prüfen ist, ob der dabei entstehende Verlust durch den erwarteten Zinssanstieg gerechtfertigt ist.

- Erwartung eines Zinsabfalls:
  Fallen die Zinsen, sollten Refinanzierungen soweit möglich aufgeschoben werden, um von den zukünftigen niedrigeren Zinsen zu profitieren. Da dies nur im Ausnahmefall möglich sein wird, muß ansonsten eine kurzfristiger orientierte Refinanzierung in kleineren Abständen durchgeführt werden.

Schon anhand dieser groben Umschreibung der Problematik läßt sich bereits die ökonomische Bedeutung einer zuverlässigen Zinsprognose erkennen. Daher wundert es nicht, daß Kreditinstitute und andere Marktteilnehmer ein großes Interesse an entsprechenden Prognoseinstrumenten haben. Gerade große Banken halten die eigenen Prognosetechniken streng geheim, da nur exklusive Informationen gewinnsteigernd eingesetzt werden können. Aus markttheoretischer Sicht ist diese Exklusivität im Hinblick auf die Problematik effizienter Märkte (siehe Kapitel 2.3.1) ebenfalls ratsam.

Das im Rahmen dieser Arbeit entwickelte FAUN-PVM Programmpaket soll zum Training Neuronaler Netze für die Zinsprognose eingesetzt werden. Grundlage sind dabei die Renditen des deutschen Pfandbriefindex PEX, für die historische Tagesdaten ab Anfang 1988 zur Verfügung standen, die durch aktuelle Daten bis Mitte Mai 2000 ergänzt wurden. Zur Berechnung des PEX werden zunächst die *Anleihezinsen* s_m für Laufzeiten von $m = 1, \ldots, 10$ Jahren ermittelt. Dazu werden zehn Pfandbriefe unterstellt, deren *Zinskupon* der laufzeitspezifischen Rendite r_m entspricht und die einen *Barwert* von 100,- DM haben. Der Anleihezins für ein Jahr ergibt sich zu $s_1 = r_1$, da die Einjahresrendite dem

Einjahresgeldmarkt entspricht. Daraus folgen die übrigen Laufzeiten rekursiv zu

$$s_m = \left(\frac{1 + r_m}{1 - r_m \cdot \left(\sum_{i=1}^{m-1} \frac{1}{(1+s_i)^i} \right)} - 1 \right)^{\frac{1}{m}} . \tag{24}$$

Die theoretischen Preise P_{mk} von 30 fiktiven synthetischen Pfandbriefen mit Restlaufzeiten von $m = 1, \ldots 10$ Jahren, den Kupons $C_k = 6$; 7,5 und 9% und dem *Nominalwert* N werden berechnet:

$$P_{mk} = \sum_{i=1}^{m} \frac{C_k}{(1+s_i)^i} + \frac{N}{(1+s_m)^m} . \tag{25}$$

Diese 30 Pfandbriefpreise werden über die Matrix Q_{mk} gewichtet und zum PEX Gesamtindex addiert:

$$\text{PEX} = \sum_{m=1}^{10} \sum_{k=1}^{3} P_{mk} \cdot Q_{mk} , \tag{26}$$

$$\text{mit } Q_{mk} = \begin{pmatrix} 3,10 & 3,50 & 4,06 & 4,88 & 4,87 & 4,09 & 3,82 & 3,38 & 3,65 & 3,15 \\ 1,73 & 2,43 & 3,03 & 3,37 & 3,15 & 2,84 & 3,02 & 3,14 & 2,62 & 1,47 \\ 2,56 & 2,87 & 3,16 & 3,70 & 4,02 & 4,32 & 4,79 & 4,06 & 3,38 & 1,84 \end{pmatrix}^T . \tag{27}$$

Die Subindizes für die Restlaufzeiten von $m = 1, \ldots, 10$ Jahren berechnen sich zu:

$$\text{PEXm} = \frac{\sum_{k=1}^{3} P_{mk} \cdot Q_{mk}}{\sum_{k=1}^{3} Q_{mk}} . \tag{28}$$

Die Kupons für die Subindizes sind die gemäß Q_{mk} gewichteten Mittelwerte von C_k =6; 7,5 und 9%. Für eine Laufzeit von einem Jahr ergibt sich z.B.:

$$(3,10 \cdot 6\% + 1,73 \cdot 7,5\% + 2,56 \cdot 9\%) : (3,10 + 1,73 + 2,56) = 7,39\% .$$

Die Renditen PEX-R m können daher aus den gegebenen Kursen mit folgenden Zinssätzen berechnet werden:

Laufzeit[Jahre]	1	2	3	4	5	6	7	8	9	10
Zinssatz[%]	7,39	7,39	7,37	7,35	7,39	7,53	7,63	7,60	7,46	7,20

In modernen Tabellenkalkulations- und Mathematikprogrammen können finanzmathematische Funktionen aktiviert werden, die eine schnelle und einfache Umrechnung des Kurses in die entsprechende Rendite ermöglichen. Eine grafische Darstellung der historischen Pfandbriefrenditen über die Laufzeiten ist in Abbildung 2 auf S. 3 zu sehen. Die Angaben

zur Berechnung des Pfandbriefindex PEX sind [DtBörseAG1] und [DtBörseAG2] entnommen, nähere Erklärungen der finanzmathematischen Begriffe finden sich in [Pfeifer].

Die Ausgaben der zu trainierenden Netze sollen der Renditezins in drei, sechs und zwölf Monaten sein, so daß hier drei mittelfristige Prognosehorizonte vorliegen. Um die Zinsstruktur über die Laufzeiten abbilden zu können, wurden die PEX-Renditen für Laufzeiten von einem, vier und zehn Jahren berücksichtigt, aus dem die fehlenden Werte quadratisch interpoliert werden können. Die mittlere Laufzeit wurde dabei näher am „kurzen Ende" gewählt, da sich hier höherfrequente ökonomische Einflüsse, z.B. Zentralbankentscheidungen, stärker niederschlagen.

Bezüglich der Netztopologie zeigten sich während der Berechnungen der unterschiedlichen Modelle einige Gemeinsamkeiten:

- Je mehr verdeckte Neuronen benutzt werden, umso schwieriger wird es, bessere Netze als bei kleinerer Topologie zu finden, d.h. um so mehr Netze müssen trainiert werden.
- Ab fünf verdeckten Neuronen werden die Netze kaum besser, manchmal steigt bei der nächst größeren Topologie der Fehler sogar an.[1]
- Keine Topologie kann sich im Vergleich zu den anderen durch einen besonders niedrigen Fehler auszeichnen.
- Einzelne Topologien zeigen starke Schwankungen in der Anpassung an den Stützzeitraum, d.h. haben teils sehr gute Phasen, teils extrem schlechte.

Diese Erfahrungen flossen in das Training und die Prognoseanwendung ein, indem einerseits bei größerer Topologie mehr Netze trainiert wurden (100 Netze pro verdecktem Neuron) um eine weitere Fehlerverminderung zu gewährleisten und andererseits durch Mittelwertbildung eine *Expert-council*[2]*-Topologie* aus den besten drei bis vier Netzen von einem bis fünf Neuronen mit und ohne Shortcuts gebildet wurde. Aufgrund der oben genannten Erfahrungen machte die Berechnung größerer Topologien wegen der in zweifacher Weise steigenden Rechenzeit (mehr Netze, die jeweils länger trainiert werden) bei minimaler oder gar keiner Verbesserung keinen Sinn. Die Verwendung einer XPC-Topologie ist auch durch die unterschiedlichen Approximationsfähigkeiten verschiedener Topologien angeraten. Netze mit wenigen Gewichten repräsentieren eine vergleichsweise einfache

[1] Theoretisch muß ein Netz mit mehr verdeckten Neuronen mindestens so gut wie eines mit weniger sein, da durch Nullsetzen der zusätzlichen Gewichte die weniger komplexe Topologie in der größeren enthalten ist.

[2] engl.: Expertenrunde; kurz XPC

Netzfunktion, die recht „steif" ist und sich den gegebenen Daten nur bedingt anpassen läßt. Daher haben Netze mit weniger verdeckten Neuronen in der Regel auch höhere Fehler als Netze mit einer größeren verdeckten Schicht. Andererseits verhindert gerade diese Steifigkeit eine Überanpassung und gewährleistet in gewissem Rahmen die Generalisierungsfähigkeit. Netze mit mehr Gewichten und einer komplizierteren Netzfunktion sind dagegen flexibler und können sich prinzipiell leichter an die vorgegebenen Daten anpassen, was aber auf der anderen Seite zu stärker oszillierenden Funktionen führt. Diese phasenweise erratischen Oszillationen sind unerwünscht und nicht nur bei Prognoseanwendungen Neuronaler Netze völlig unbrauchbar. Durch Bildung einer XPC-Topologie können die positiven Effekte – Generalisierung bei guter Anpassung – hervorgehoben werden, während allzu starke Oszillationen gedämpft werden. Da trotz der XPC-Topologie noch hochfrequente Oszillationen vorhanden waren, wurde die Prognose nochmal mit einem gewichtetem gleitendem Durchschnitt über 21 Tage (siehe 4.2.2) geglättet.

4.2 Modellauswahl

Die schwierigste Frage der am besten geeigneten Eingabedaten mußte weitgehend experimentell ermittelt werden, indem eine Vielzahl unterschiedlicher Modelle berechnet und ausgewertet wurde. Gemeinsam war allen Modellen, daß eine *unbedingte Prognose* durchgeführt wurde, bei der zukünftige Werte durch zeitlich vorhergehende unabhängige Variablen ermittelt werden, die zeitverzögert in das Modell eingehen. Bei einer bedingten Prognose wird dagegen eine Zielgröße zum Zeitpunkt t als abhängig von anderen unabhängigen Variablen zur Zeit t angenommen, so daß für die Prognose der Zielgröße zunächst die Werte der unabhängigen Variablen prognostiziert werden müssen. Daher sind bedingte Prognosemodelle vorzuziehen, da im anderen Fall nur das Problem einer zu prognostizierenden Größe durch mehrere Größen ersetzt wird. Siehe dazu auch [Poddig, S. 221]. In den hier vorzustellenden Modellen wurden stets Eingabewerte verwendet, die maximal in der Größe des Prognosehorizonts in der Vergangenheit lagen.

Insgesamt wurden 14 verschiedene Modelle getestet, aus denen das vielversprechendste ausgesucht wurde. Um den enormen Aufwand zu minimieren und im zur Verfügung stehenden zeitlichen Rahmen zu belassen, wurden zunächst nur die Ergebnisse für die Dreimonatsprognose der PEX-Rendite für die Laufzeit von einem Jahr verglichen. Dahinter steht die Annahme, daß die Modelle im wesentlichen von der Auswahl der Eingabegrößen abhängen und die Bewertung daher vom konkreten Prognosehorizont und der Restlaufzeit weitgehend unabhängig ist, solange identische Konstellationen verglichen werden. Tatsächlich konnte das ausgewählte Modell auch bei den anderen Prognosehorizonten und Laufzeiten vergleichbar gute Ergebnisse erzielen.

Die betrachteten Modelle lassen sich grob in vier Gruppen aufteilen, einmal anhängig vom Charakter der Eingabedaten und dann unterteilt nach Delta- und Punktprognose. Zunächst wurden drei relativ einfache Modelle betrachtet, bei denen die Eingabewerte des Netzes unterschiedliche rückwärtige und laufzeitübergreifende Differenzen aus dem Renditegebirge darstellen, während die Ausgabe der gesuchten Kursdifferenz in drei Monaten entspricht (Deltaprognose). In einem Fall wurden zum Vergleich zurückliegende Werte direkt eingegeben und eine Punktprognose durchgeführt. In einem zweiten Ansatz lagen den Eingabewerte im wesentlichen *Technische Indikatoren* zugrunde, wie sie heutzutage z.B. in der Aktienkursprognose immer häufiger verwendet werden. Zur besseren Orientierung sind die Modelle nach folgendem Schema bezeichnet, mit dem sich auch die im Anhang angegebenen Modelle zuordnen lassen:

	Prognoseart	
Eingabegrößen	Delta (a)	Punkt (b)
Differenz-/Punktwerte (I)	Ia1, Ia2, Ia3	Ib1
Indikatoren (II)	IIa1, IIa2, IIa3, IIa4, IIa5	IIb1, IIb2, IIb3, IIb4, II5

Im folgenden sollen die wesentlichen Ergebnisse anhand der wichtigsten Modelle vorgestellt werden. Die statistischen Werte und Grafiken der nicht explizit erwähnten Modelle finden sich im Anhang ab S. 136. Zunächst werden die Art und Anzahl der Eingabegrößen, der verwendete Vergleichsfaktor und die Aufteilung der Trainings- und Validierungsmuster angegeben. Dann folgen Angaben, welche die Trainingsergebnisse abhängig von der Topologie darstellen, insbesondere Trainings- und Validierungsfehler, Anzahl initialisierter Netze und Rechenzeiten. Nach dem Training wurde eine weitergehende Untersuchung der Prognosegüte durchgeführt, bei der die Trefferquoten einer reinen steigt/fällt-Prognose, der mittlere absolute Fehler, der mittlere quadratische Fehler, die Korrelation zwischen Prognose und tatsächlichem Wert und der Theil'sche Ungleichheitskoeffizient berechnet wurden. Im ersten Beispiel werden zudem die Werte einer linearen Regression basierend auf den drei Vormonaten und der naiven Delta- und Punktprognose angegeben. Bei der Deltaprognose entspricht die naive Prognose nicht dem aktuellen Wert, sondern analog der letzten Änderung bezogen auf den Prognosehorizont. Schon mit diesem recht einfachen Modell konnten die naiven Prognosen und die lineare Regression eindeutig geschlagen und die Überlegenheit Neuronaler Netze gezeigt werden. Letztendlich wurde aber den Indikatorenmodellen der Vorzug gegeben, da bei den Differenz- und Punktwertmodellen nur das Modell mit dem stark geglättetem Basiswert überzeugen konnte. Durch den zentrierten Gleitenden Durchschnitt über 63 Tage kann knapp die Hälfte dieses Zeitraums (31 Tage), der auf diese Weise nicht mehr geglättet werden kann, im Prognosehorizont nicht mehr sinnvoll berücksichtigt werden. Bei der hier durchgeführten mittelfristigen Prognose auf drei bis zwölf Monate konnte dies nicht toleriert werden.

Alle Werte wurden für den verwendeten Stützzeitraum der Trainings- und Validierungsdaten und für einen Generalisierungszeitraum von drei Monaten ermittelt. Da hier zum Zeitpunkt der Modellauswahl nur eine recht kleiner Generalisierungszeitraum mit einem eindeutigen Trend zur Verfügung stand, sind die teilweise sehr guten Ergebnisse, besonders der Trefferquoten, vorsichtig zu bewerten. Beim endgültig ausgewählten Modell ist ein längerer Generalisierungszeitraum berücksichtigt, der beim dreimonatigen Prognosehorizont ca. ein halbes Jahr, beim sechsmonatigen Horizont neun Monate und bei der Einjahresprognose 18 Monate betrug. Bei den Tabellen wurden folgende Abkürzungen benutzt:

n_i	Anzahl Eingabegrößen
F	Vergleichsfaktor zwischen Trainings- und Validierungsdatenfehler zur Vermeidung von Übertraining
A.T.M.	Anzahl Trainingsmuster
A.V.M.	Anzahl Validierungsmuster
n_h	Anzahl verdeckter Neuronen
s.c.	Shortcuts
A.G.	Anzahl trainierbarer Gewichte
A.I.N.	Anzahl initialisierter Netze bis zum erfolgreichen Training
A.T.N.	Anzahl trainierter Netze
Verh.	Verhältnis initialisierter zu trainierten Netzen
$\emptyset\, t_{e.t.N.}$	Durchschnittliche Trainingszeit pro erfolgreich trainiertem Netz
ε_t^*	bester Trainingsdatenfehler
ε_v^*	bester Validierungsdatenfehler
$\emptyset\, \varepsilon_t^*$	durchschnittlicher bester Trainingsdatenfehler (bezogen auf A.T.M.)
$\emptyset\, \varepsilon_v^*$	durchschnittlicher bester Validierungsdatenfehler (bezogen auf A.V.M.)
R.T.Q.	Trefferquote der Richtungsprognose (steigt/fällt)
M.A.F.	mittlerer absoluter Fehler
M.Q.F.	mittlerer quadratischer Fehler
r	Bravais-Pearson-Korrelationkoeffizient
R^2	Bestimmtheitskoeffizient (R^2=r^2)
U_d, U_p	Theil´scher Ungleichheitskoeffizient (auf Basis der naiven Delta- bzw. Punktprognose)

Bei den grafischen Darstellungen ist auf der Zeitachse die interne Numerierung von Handelstagen aufgezeichnet, die am Freitag, den 2.1.1987 beginnt. Dementsprechend wird hier der Handelstag 251 für Montag, den 4.1.1988 als erstes aufgeführt. Zwischen zwei Gitternetzlinien der Zeitachse liegen hier 63 Handelstage oder ungefähr drei Monate. Die

Modell	Eingabegrößen
Ia1	Rückwärtsdifferenzen von PEX1-R und PEX10-R auf 1, 2 und 3 Monate Differenzen zwischen PEX1-R und PEX10-R heute und vor 1, 2 und 3 Monaten
Ia2	wie Ia1, nur auf Basis eines zentrierten gleitenden Durchschnitts über 21 Handelstage (entspricht ca. einem Monat)
Ia3	wie Ia2, allerdings mit 63 Tagen
Ib1	PEX1-R und PEX10-R heute und vor 1, 2, 3 Monaten

Tabelle 7: Verwendete Eingabegrößen der Differenz- und Punktwertmodelle

vergrößerten Darstellungen beziehen sich auf die letzten neun Monate des hier betrachteten Zeitraums vom 26.3.99 bis zum 22.12.1999, mit jeweils 21 Handelstagen (ca. ein Monat) zwischen den Linien. Die Zahlenwerte der y-Achse bezeichnen die Zinssätze in Prozent.

4.2.1 Differenz- und Punktwertmodelle

Die Eingabegrößen der Modelle sind in Tabelle 7 zusammengefaßt. Vorgestellt wird nur das Modell Ia3, daß in dieser Gruppe am besten abgeschnitten hat, was sicher auf die starke Glättung des Basiswerts zurückzuführen ist. Zur Verifizierung finden sich die Ergebnisse der übrigen Modelle im Anhang ab S. **??**. Zum Vergleich sind noch die statistischen Werte der linearen Regressions-, sowie der naiven Delta- und Punktprognose auf Basis der ungeglätteten PEX-Rendite angegeben, wobei erstere auch grafisch dargestellt ist. Da die naive Deltaprognose der linearen Regression sehr ähnlich, nur weniger glatt ist, wurde diese nicht extra aufgeführt. Die ebenfalls nicht dargestellte naive Punktprognose entspricht dem Basiswert, der um den Prognosehorizont nach rechts in die Zukunft verschoben ist. Bei der Berechnung von U wurde stets der Basiswert des jeweiligen Prognosemodells zugrundegelegt, sprich die entsprechende Glättung. Die statistischen Werte zur Prognosegüte weichen bei den drei Vergleichsmodellen abhängig von der Glättung und der Prognoseart allerdings kaum voneinander ab. Bei der Betrachtung dieser ersten Klasse von Modellen fallen bereits zwei wichtige Ergebnisse auf, die sich auch bei den Indikatormodellen finden. Zum einen neigen alle betrachteten Modelle dazu, zumindest in gewissen Phasen eine naive Punkt-Prognose abzugeben, was interessanterweise auch für die Deltaprognosemodelle gilt. Daher ist zu hinterfragen, ob die Berechnung des Ungleichheitskoeffizienten U nicht generell auf Basis der naiven Punktprognose erfolgen sollte, um einen sinnvollen Vergleich der Prognosegüte durchführen zu können. Zum Vergleich sind hier beide Varianten angegeben. Die Tendenz der Modelle zur naiven Punktprognose unabhängig von den Eingabegrößen und anderen Parametern könnte von Vertretern der Random Walk Hypothese als deren Bestätigung angesehen werden. Dennoch konnten durch intensive Arbeit an den Modellen die anfänglich extrem starken naiven Anteile auf ein akzeptables Maß reduziert werden, was durch die relativ geringen Werte für U bestätigt wird. Die Reduktion des naiven Anteils der Prognose sollte vorrangiges Ziel weiterer Untersuchungen sein. Das einzige Punktprognosemodell in dieser Klasse konnte im Vergleich zu den drei Deltaprognosemodellen nicht überzeugen. Bei den Indikatormodellen wurden systematisch für jedes Modell Delta- und Punktprognosen durchgeführt, die dieses Ergebnis bestätigen. Als negatives Beispiel einer Punktprognose ist daher allein das Modell IIb1 in 4.2.2 angegeben. Abschließend sei noch erwähnt, daß Versuche weiter in die Vergangenheit zurückzugreifen oder einen kürzeren historischen Stützbereich zu verwenden, keine Verbesserung der Prognose zur Folge hatte. Bei der Suche nach besseren Modellen wird daher die Frage der geeigneten Eingabegrößen und der Art ihrer Verwendung im Vordergrund stehen müssen.

Modell Ia3

- Rückwärtsdifferenzen von PEX-R 1 und PEX-R 10 auf 1, 2 und 3 Monate
  Differenzen zwischen PEX-R 1 und PEX-R 10 heute und vor 1, 2 und 3 Monaten auf Basis eines zentrierten gleitenden Durchschnitts über 63 Handelstage (zGD63) (entspricht ca. drei Monaten)

n_i	10	A.T.M.	2254	Anteil V.M.	20,10%
F	1,20	A.V.M.	567	Aufteilung T/V	225/63/...

n_h	1		2		3		4		5	
s.c.	n	j	n	j	n	j	n	j	n	j
A.G.	13	23	25	35	37	47	49	59	61	71
A.I.N.	3716	272	3705	1176	6216	3536	9377	5642	12159	8066
A.T.N.	100	100	200	200	300	300	400	400	500	500
Verh.	37,16	2,72	18,43	5,88	20,72	11,79	23,44	14,11	24,32	16,07
$\emptyset\, t_{e.t.N.}$	1,19	1,47	11,00	3,30	16,51	8,24	21,38	6,18	18,41	9,09
ε_t^*	84,35	41,75	41,55	36,05	37,18	29,02	29,80	28,06	25,88	24,26
ε_v^*	78,87	48,43	49,68	43,01	40,94	28,30	27,89	28,52	30,99	28,89
$\emptyset\, \varepsilon_t^*$	0,261	0,184	0,183	0,171	0,173	0,153	0,155	0,150	0,144	0,140
$\emptyset\, \varepsilon_v^*$	0,252	0,198	0,200	0,186	0,182	0,151	0,150	0,152	0,158	0,153

Tabelle 8: Trainingsergebnisse der Prognose von PEX-R 1 auf 3 Monate

	Stützzeitraum				Generalisierung			
Modell	Ia3	linear	$naiv_d$	$naiv_p$	Ia3	linear	$naiv_d$	$naiv_p$
R.T.Q.	86,42%	54,34%	55,88%	-	100,00%	62,77%	62,77%	-
M.A.F.	0,148	0,571	0,476	0,486	0,150	0,488	0,368	0,375
M.Q.F.	0,043	0,629	0,428	0,460	0,027	0,412	0,251	0,235
r	0,995	0,945	0,958	0,969	0,968	0,927	0,933	0,919
R^2	0,990	0,892	0,918	0,940	0,936	0,859	0,871	0,844
U_d	0,420	1,213	-	-	0,388	1,281	-	-
U_p	0,394	1,169	-	-	0,392	1,324	-	-

Tabelle 9: Prognosegüten von Ia3 im Vergleich zur Prognose basierend auf der linearen Regression der drei Vormonate sowie der naiven Delta- und Punktprognose.

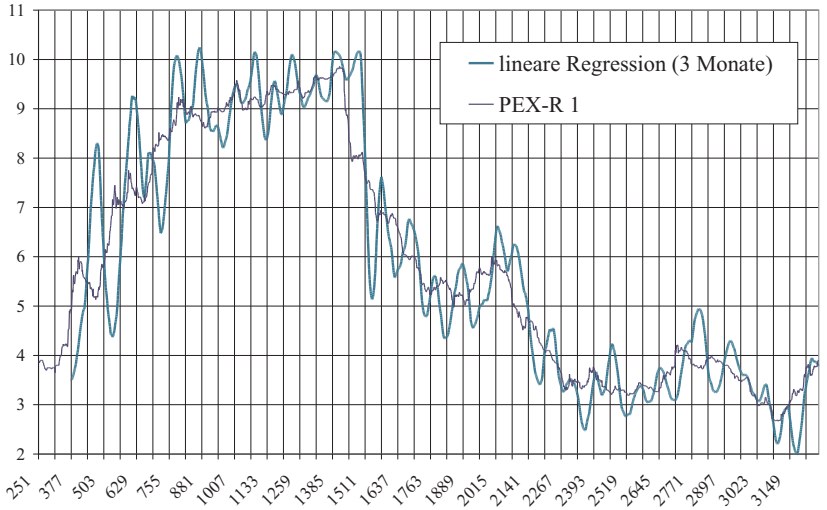


Abbildung 13: Prognose durch lineare Regression auf Basis der jeweils letzten 3 Monate

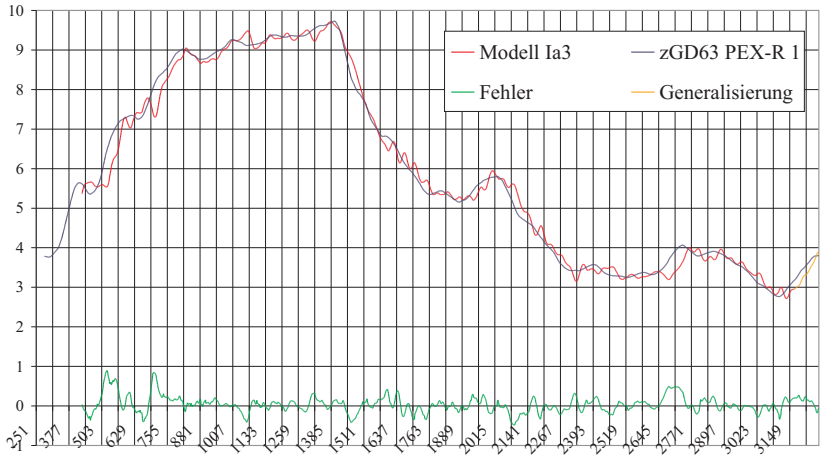


Abbildung 14: Deltaprognose durch Differenzenmodell Ia3 auf stark geglättetem (zGD63) Basiswert

4.2.2 Indikatorenmodelle

In einem zweiten Schritt wurden Modelle untersucht, in die Technische Indikatoren als Eingabegrößen einfließen, welche in den letzten Jahren immer bekannter und beliebter geworden sind. Diese Indikatoren sind aus der Zeitreihe abgeleitete Größen, wobei nicht nur der Wert an sich, sondern vor allem seine Entwicklung im Vordergrund steht. Indikatoren dienen in der Mehrzahl der Fälle dazu Handelssignale zu geben, sobald eine Trendumkehr erkannt wird. Diese Signale entstehen meistens durch Schneiden von Mittelpunkts- oder Extremwertlinien. Da Technische Indikatoren von Praktikern aus der Ökonomie entworfen wurden, sind diese fast immer rein heuristisch durch persönliche Erfahrungen und Vorstellungen entstanden, ohne nach einem theoretischen Fundament ihrer Wirkungsweise zu fragen. Dennoch können Indikatoren nachweislich gute Handelssignale liefern und Kursprognosen stützen.

Die beiden wichtigsten Gruppen von Indikatoren sind die *Trendfolger* und die *Oszillatoren*, die in unterschiedlichen Phasen von Marktentwicklungen sinnvoll einzusetzen sind. Trendfolger sollen die längerfristige Kursrichtung und deren Stärke anzeigen, während Oszillatoren in trendarmen Phasen, den sogenannten Seitwärtsbewegungen, vernünftige Signale erzeugen können. Da ein einsetzender Trend erst erkannt werden muß, kann diesem nur mit Verzögerung gefolgt werden, d.h. Handelssignale können nicht im günstigsten Augenblick (Tiefst-/Höchstkurs) erzeugt werden, sondern etwas verspätet. Dieses Prinzip der Informationssammlung und verzögerter Signalerzeugung liegt grundsätzlich bei allen Indikatoren vor. Zwar können durch Parametereinstellungen und bestimmte Techniken kürzere Reaktionszeiten ermöglicht werden, allerdings immer zum Preis eines erhöhten Anteils von Fehlsignalen. Erfolgreich arbeiten die Trendfolger nur in Phasen andauernder und nachhaltiger Trends, bei Seitwärtsbewegungen des Marktes häufen sich Fehlsignale. Ebenso werden wegen der verzögerten Signalerzeugung Übertreibungen des Marktes nach oben oder unten, die eine entgegengesetzte Trendwende vermuten lassen, nicht angezeigt. Die Basis der Trendfolger stellen die gleitenden Durchschnitte dar, aus denen viele andere Indikatoren durch Abwandlungen hervorgehen.

Die zweite große Gruppe bilden die Oszillatoren, die im allgemeinen zwischen einer oberen und unteren Linie pendeln und ggf. eine Mittelpunktslinie kreuzen. Werden die Extremlinien erreicht, wird eine überkaufte oder -verkaufte Situation, d.h. Marktübertreibungen, angezeigt. Daraus folgt dann ein Verkaufssignal bei noch bestehender Aufwärts- und ein Kaufsignal bei Abwärtsbewegung. Oszillatoren funktionieren gut bei Seitwärtsbewegungen des Marktes ohne eindeutigen und anhaltenden Trendeinfluß, versagen aber sobald ein solcher vorliegt. Die Grundlage der Oszillatoren bildet das *Momentum*, daß in seiner einfachsten Form die Differenz aufeinanderfolgender Kurse ist und somit die diskrete Ableitung

einer Zeitreihe darstellt. Daneben gibt es noch weitere, weniger bedeutende Gruppen, wie die *Trendintensitätsindikatoren*, die Trendphasen identifizieren und bewerten sollen, die *Umsatzindikatoren*, die Volumentrends und -wechsel anzeigen und die *Volatilitätsindikatoren*, die Maßzahlen für die „Beweglichkeit" der Kurse angeben. Diese Indikatoren werden kaum eigenständig, sondern fast immer in Kombination mit Trendfolgern und Oszillatoren eingesetzt. Eine sehr ausführliche Darstellung zur Zeit bekannter und verwendeter Indikatoren findet sich in [Müller].

Anhand der vorliegenden Literatur[3] und durch eigene Betrachtungen zur Güte der Handelssignale einiger Indikatoren kristallisierten sich zunächst mehrere Indikatoren heraus, die für einen Einsatz zur PEX-Prognose geeignet schienen:

Trendfolger:	Gleitende Durchschnitte (kurz GDs) und Differenzen davon
Oszillatoren:	MACD, RSI, Coppock, Stochastik, Momentum, TRSI
Trendintensität:	ADX, TBI

Diese Auswahl wurde anhand zweier Kriterien weiter verkleinert, einmal der Redundanz der Indikatoren untereinander und dem Aufwand und Komplexität ihrer Berechnung. Letzteres ist vor allem im Hinblick auf die spätere praktische Nutzung eines Prognosemodells wichtig, das in diesem Fall innerhalb einer Tabellenkalkulation implementiert wurde. Durch die zur Verfügung stehenden langen Zeitreihen entstehen letztlich sehr große Datenmengen[4] durch Berechnung der Indikatoren und dem Bereithalten der jeweils skalierten und unskalierten Werte. Selbst auf den heutigen schnellen Rechnern mit großem Hauptspeicher kann es so zu Beeinträchtigungen bei der Benutzung kommen. Aufgrund von Korrelationsuntersuchungen Technischer Indikatoren am Beispiel des DAX-Future aus [Wittmer] fiel die weitere Wahl auf den MACD und RSI, während Momentum und Stochastik[5] verworfen wurden. Der TBI wurde ebenfalls aus der Auswahl entfernt, da seine Aussage als Quotient zweier GDs auch durch die entsprechende Differenz ausgedrückt werden kann. Da hier die Handelsignale bei gleichen Werten der GDs entstehen, ist dies im einen Fall beim Wert 1 im anderen beim Wert 0 der Fall. Ebenso entfiel der MACD, der durch den Schnitt dreier GDs entsteht und daher ebenfalls keine neuen Informationen liefert. Da der Coppock-Indikator für Prognosen über einem Jahr ausgelegt ist, wurde auf seinen Einsatz für die mittelfristige Prognose verzichtet. Der ADX entfiel wegen seiner aufwendigen Berechnungsmethode, so daß schließlich nur noch Gleitende Durchschnitte und ihre Differenzen, RSI und TRSI als Eingabegrößen für das Neuronale Netz in Betracht kamen.

[3] vor allem [Müller], [Wittmer], [NordLB1] und [NordLB2]

[4] Für jeden Prognosehorizont entstand eine Datei von über 30 Megabyte.

[5] Nicht zu verwechseln mit der gleichnamigen mathematischen Disziplin, mit der dieser Indikator trotz seines Namens nicht viel zu tun hat.

Diese Auswahl mag einem auf den ersten Blick recht klein vorkommen, doch ist zu bedenken, daß viele Indikatoren nur Abwandlungen voneinander sind, die auf den gleichen Grundlagen beruhen und daher einen stark redundanten Informationsgehalt haben. Gerade diese Redundanz sollte aber bei der Netzeingabe vermieden werden. Daß in der Praxis wesentlich mehr Indikatoren genutzt werden, liegt u.a. an den unterschiedlichen Perspektiven, die diese dem menschlichen Betrachter bieten, obwohl ihre Aussagen und Signale vergleichbar sind. Welcher der vielen Indikatoren verwendet wird, hängt auch von den persönlichen Vorlieben und dem Glauben an seine Wirkungsweise ab. Neuronale Netze sollten dagegen aufgrund redundanzarmer Information eigenständig den Zusammenhang zwischen Indikator und Kursentwicklung finden. Wichtiger als die Wahl des Indikators ist daher die Einstellung seiner Parameter auf den betrachteten Basiswert und die gewünschten Aussagen.

Bei den folgenden Definitionen der verwendeten Indikatoren steht C für die jeweiligen Schlußkurse, t für den Zeitindex der betrachteten Reihe und T, X und Y für wählbare Parameter:

Gewichtete Gleitende Durchschnitte

Im Gegensatz zu den einfachen GDs werden hier aktuelle Kurse stärker, vergangene dagegen schwächer berücksichtigt. Der praktische Nutzen liegt darin, daß die gewichteten GDs sich nicht so weit von der ursprünglichen Kurve entfernen, wie dies bei den einfachen GDs der Fall ist, die ihrem Basiswert stärker hinterherlaufen. Für das Prognosemodell wurden exponentiell gewichtete GDs als Eingabegrößen verwendet, während die XPC-Ausgabe linear geglättet wurde, um unerwünschte Oszillationen zu dämpfen. Nach der allgemein üblichen englischen Bezeichnungsweise werden der linear gewichtete GD mit WMA (weighted moving average), und der exponentiell gewichtete mit EMA (exponential weighted moving average) bezeichnet.

$$\mathrm{WMA\,T}_t = \frac{\sum_{\tau=0}^{T-1} C_{t-\tau}\cdot(T-\tau)}{\sum_{t=1}^{T} t}, \tag{29}$$

$$\mathrm{EMA\,T}_t = \mathrm{EMA\,T}_{t-1} + \frac{2}{\mathrm{T}+1}\cdot(C_t - \mathrm{EMA\,T}_{t-1})\ . \tag{30}$$

Ihre Aussagen beziehen die GDs im wesentlichen durch Differenz- und Schnittbildung mit dem Basiswert oder weiteren GDs mit anderen Laufzeiten, bei der die Höhe der Differenz die Trendstärke angibt. Grundsätzlich gilt dabei:

kurzer GD > langem GD: Aufwärtstrend

langer GD < kurzem GD: Abwärtstrend

kurz schneidet lang von oben: Trendwechsel von auf zu ab → Verkaufen („short")

kurz schneidet lang von unten: Trendwechsel von ab zu auf → Kaufen („long")

Beim Schnitt zweier GDs sind deren Parameter von entscheidender Bedeutung. Je kleiner der Unterschied zwischen kurzem und langem GD, um so schneller und häufiger werden Signale erzeugt, unter denen dann aber auch häufiger Fehlsignale zu finden sind. Im umgekehrten Fall werden weniger, aber zuverlässigere Signale erzeugt, die mit höherer Wahrscheinlichkeit langfristige Trends anzeigen können.

Relative Stärke Index

Der RSI ist ein Kontratrend-Oszillator, der die sogenannte *innere Stärke* eines Titels anzeigt, die bei Extremwerten eine überkauft/überverkauft-Situation anzeigt. Der Name ist irreführend, denn die *relative Stärke* bezeichnet üblicherweise die Stärkerelationen zwischen mehreren Titeln, z.B. Aktienkursen. Der RSI ist eine Weiterentwicklung des Momentums und soll auf Übertreibungen hinweisen.

Zunächst werden die Auf- (u=up) und Abwärtskurse (d=down) des betrachteten Zeitraums ermittelt:

$$\begin{aligned} u = C_t,\quad d = 0 \quad &\text{falls } C_t > C_{t-1}\,,\\ u = 0,\quad d = C_t \quad &\text{falls } C_t < C_{t-1}\,. \end{aligned}$$

Dann werden die Durchschnitte U,D von u und d über den betrachteten Zeitraum T ermittelt, aus deren Quotient der RSI folgt:

$$\mathrm{RSI\,T} = 100 - \frac{100}{1+\frac{U}{D}}\,. \tag{31}$$

Der RSI oszilliert zwischen 0 (minimale innere Stärke, überverkauft) und 100 (maximale innere Stärke, überkauft). Meistens wird schon das Überschreiten der 30er bzw. 70er Linie als Verkaufs- bzw. Kaufsignal gewertet, um eine geringere Verzögerung zu erreichen und eine angemessene Anzahl von Signalen zu erhalten.

„Wahrer" Stärke Index

Der TRSI[6] basiert auf einer doppelten Glättung des Momentums, bei der einzelne Kursänderungen betrachtet werden und nicht der Quotient der Durchschnitte wie beim RSI. Mom_1 bezeichne das Eintagesmomentum, d.h. die Differenz der Schlußkurse, dann errechnet sich der TRSI zu

$$\mathrm{TRSI\,X/Y} = \frac{\mathrm{EMA\,X\,(EMA\,Y\,}(Mom_1))}{\mathrm{EMA\,X\,(EMA\,Y\,}(|Mom_1|))} \cdot 100\,. \tag{32}$$

Der Zeitparameter X wird dabei kurz, Y länger gewählt. Standardeinstellung ist 3/14 Tage, weitere Empfehlungen sind 6/20 („fast"), 20/40 („slow") und 40/80 („slower"). Der TRSI oszilliert zwischen -100 und 100 um die Nullinie, bei deren Schneiden von unten nach oben ein Kaufsignal, im umgekehrten Fall ein Verkaufssignal folgt. Diese Signale erfolgen wie üblich zeitverzögert, schnellere Reaktionen können durch Schneiden von Signallinien (z.B. -50/50) oder Gleitenden Durchschnitten erreicht werden.

Nicht nur die Wahl der Indikatoren ist wichtig, auch die konkrete Einstellung ihrer Parameter. Aufgrund eigener Beobachtungen zur Qualität und Häufigkeit der Handelssignale wurden der RSI 63 und der TRSI 10/21 für die Prognose auf drei Monate gewählt. Als Kriterium wurde die Zuverlässigkeit der historischen Signale verwendet, wobei deren Abstände ungefähr in der Größenordnung des Prognosehorizonts liegen sollten. Beim endgültigen Modell wurden die Parameter für sechs und zwölf Monate entsprechend verdoppelt bzw. vervierfacht. Als zu prognostizierendem Wert wurde ein mit dem EMA 5 oder EMA 10 geglätteter PEX-R benutzt. Als Eingabegrößen kamen dann noch die Differenzen zu drei weiteren exponentiell geglätteten GDs des PEX-R hinzu. Für die Dreimonatsprognose wurde die engere EMA 21/42/63 und die weiter auseinanderliegende EMA 21/63/126 Kombination verwendet, mit entsprechenden Vielfachen für andere Prognosehorizonte. Dadurch sollte im nachhinein festgestellt werden, ob eher die kurzfristig reagierenden oder zuverlässigere, aber stärker verzögerte Signale für die Prognose geeignet sind. Die beschriebenen Indikatoren wurden auf drei verschiedene Arten als Eingabegrößen genutzt:

- direkt, ohne weitere Transformationen (Modell 1)
- direkt, zuzüglich des jeweiligen Eintagesmomentum (2,3)
- transformiert in Signalgrößen (4,5)

[6] **TR**ue **S**trength **I**ndex: Die übertriebene Namensgebung offenbart den großen individuellen Einfluß bei der Erfindung eines Indikators.

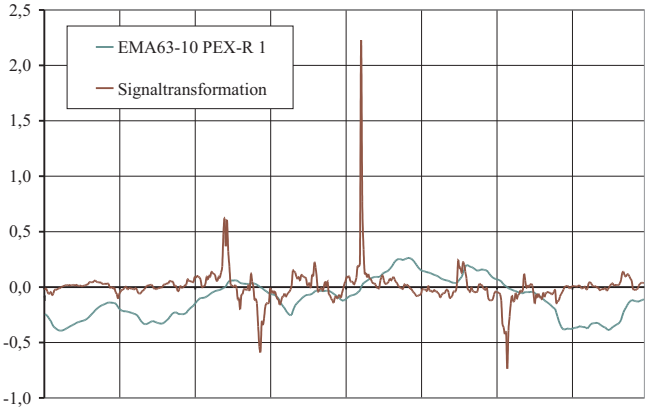


Abbildung 15: Transformierte Eingabewerte der Differenz von EMA63 und EMA10 des PEX-R 1, durch die Signalsituationen erfaßt werden. Schneiden sich die Gleitenden Durchschnitte kreuzt die Differenz die Nullinie und es wird ein Wert abhängig von der Steigung (Eintagesmomentum) erzeugt.

Die erste Variante sollte zeigen, ob eine direkte Nutzung der Indikatoren, so wie es ein menschlicher Anwender tun würde, überhaupt sinnvoll mit Neuronalen Netzen möglich ist. Bei der zweiten Möglichkeit sollte getestet werden, ob die Kombination mit dem Eintagesmomentum als diskrete Ableitung bessere Ergebnisse erzielt. Dahinter stand die Hoffnung, daß ein Neuronales Netz durch diese zusätzliche Information erkennt, in welcher Richtung und mit welcher Steigung etwaige Linien bzw. GDs gekreuzt werden. Die dritte Variante transformierte die Indikatoren so, daß in signalgebenden Situationen große Werte erzeugt wurden, während die Eingabe ansonsten nahe oder gleich Null war (siehe Abb. 15). So wurde die Differenz Gleitender Durchschnitte und der TRSI durch den entsprechenden Reziprokwert gewichtet mit dem Eintagesmomentum ersetzt, der RSI bei Schneiden der 30er oder 70er Linie durch sein Eintagesmomentum. Die Eingabegrößen der Indikatorenmodellen sind Tabelle 10 zu entnehmen.

Vorgestellt werden zunächst die Modelle IIa1 und IIb1, in welche die Indikatoren direkt eingeflossen sind. Hier ist insbesondere der Vergleich von Delta- und Punktprognose interessant, der ein vollkommen unbrauchbares Ergebnis der Punktprognose ergibt. Dies gilt analog für alle anderen Modelle, so daß ansonsten keine weitere Punktprognose mehr betrachtet wird. Zu beachten sind die teils langen Phasen eines konstanten Prognosewertes, sowie der große Sprung in der Mitte des Stützzeitraums, der aus dem Nichts zu kommen scheint. Durch die andersgeartete Wertemenge der Punktprognose operiert die Netzfunktion bei extremen Werten im Grenzbereich der Tangenshyperbolicus-Funktion,

Modell	PEX-R 1	PEX-R 10
1	EMA 5, 10-5, 21-5, 63-5, RSI63, TRSI 10/21	EMA 5, 10-5, 21-5, 63-5, RSI63, TRSI 10/21
2	EMA 10; EMA 21-10, 42-10, 63-10, RSI63, TRSI 10/21 jew. mit Mom_1	EMA 10, EMA42-10 mit Mom_1
3	EMA 10; EMA 21-10, 63-10, 126-10, RSI63, TRSI 10/21 jew. mit Mom_1	EMA 10, EMA63-10 mit Mom_1
4	EMA 10; EMA 21-10, 42-10, 63-10 jeweils mit Signaltransformation, RSI63, TRSI 10/21 nur Signaltransf.	EMA 10, EMA42-10 mit Signaltransf.
5	EMA 10, 21-10, 63-10, 126-10 jeweils mit Signaltransformation, RSI63, TRSI 10/21 nur Signaltransf.	EMA 10, EMA63-10 mit Signaltransf.

Tabelle 10: Verwendete Eingabegrößen der Indikatorenmodelle für die Delta- und Punktprognose (Klassen IIa bzw. IIb).

wodurch sich die konstanten Phasen erklären. Im Vergleich zur Deltaprognose ist diese Art der Prognose dementsprechend zu verwerfen.

Die besten Modelle, IIa3 und IIa4, kamen schließlich in die engere Auswahl des zu bestimmenden endgültigen Modells. Interessant ist, daß sie sich sowohl in der Kombination der GD-Differenzen, als auch in den zusätzlichen Eingabegrößen unterscheiden, so daß hier keine eindeutige Aussage über die geeignetesten Eingabegrößen getroffen werden kann. Zu diesen beiden Modellen ist auch eine vergrößerte Darstellung der letzten neun Monate des betrachteten Zeitraum angegeben.

Modelle IIa1

- EMA 5, EMA 10-5, EMA 21-5, EMA 63-5, RSI63, TRSI 21/10
  von PEX-R 1 und 10

n_i	12	A.T.M.	2255	Anteil V.M.	20,09%
F	1,05	A.V.M.	567	Aufteilung T/V	223/63/...

n_h	1		2		3		4		5	
s.c.	n	j	n	j	n	j	n	j	n	j
A.G.	15	27	29	41	43	55	57	69	71	83
A.I.N.	337	12214	1264	16202	2443	18224	3732	17224	4991	15601
A.T.N.	100	100	200	200	300	300	400	400	500	500
Verh.	3,37	122,14	6,32	81,01	8,14	60,75	9,33	43,06	9,96	41,38
$\emptyset\, t_{e.t.N.}$	0,54	51,65	3,55	43,76	4,66	40,20	6,59	32,47	9,44	45,87
ε_t^*	70,73	55,86	53,85	46,59	50,50	42,22	43,04	52,92	44,29	59,08
ε_v^*	73,09	65,17	53,95	54,17	39,80	47,58	43,09	58,42	44,33	70,06
$\emptyset\, \varepsilon_t^*$	0,250	0,223	0,219	0,203	0,212	0,194	0,195	0,217	0,198	0,229
$\emptyset\, \varepsilon_v^*$	0,255	0,240	0,219	0,219	0,188	0,205	0,195	0,228	0,198	0,249

Tabelle 11: Trainingsergebnisse der Deltaprognose IIa1 von PEX-R 1 auf 3 Monate

	Stützbereich		Generalisierung	
Modell	IIa1	IIb1	IIa1	IIb1
R.T.Q.	77,62%	72,69%	67,02%	67,02%
M.A.F.	0,261	0,300	0,155	0,174
M.Q.F.	0,119	0,158	0,032	0,045
r	0,989	0,985	0,904	0,791
R^2	0,978	0,971	0,817	0,625
U_d	0,498	0,328	0,361	0,426
U_p	0,609	0,490	0,350	0,413

Tabelle 12: Prognosegüten der Deltaprognose IIa1 und der analogen Punktprognose IIb1 im Vergleich. Trotz ähnlicher Werte ist die Deltaprognose eindeutig vorzuziehen, wie umstehender Abbildung zu entnehmen ist.

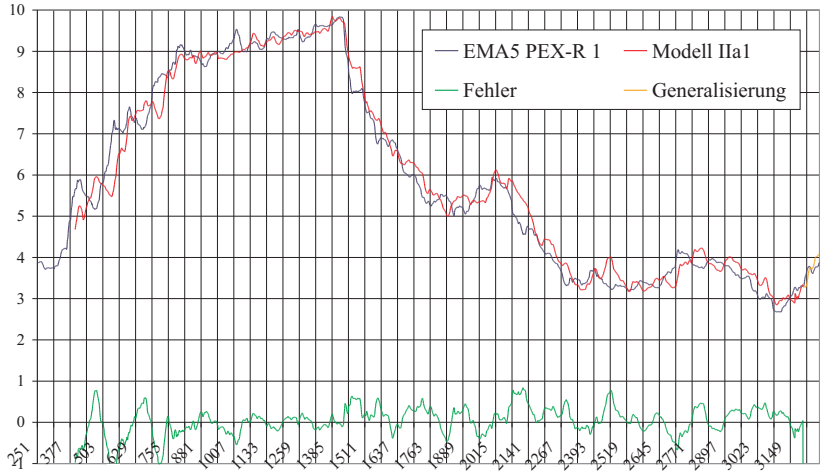


Abbildung 16: Deltaprognose von PEX-R 1 auf 3 Monate durch Modell IIa1

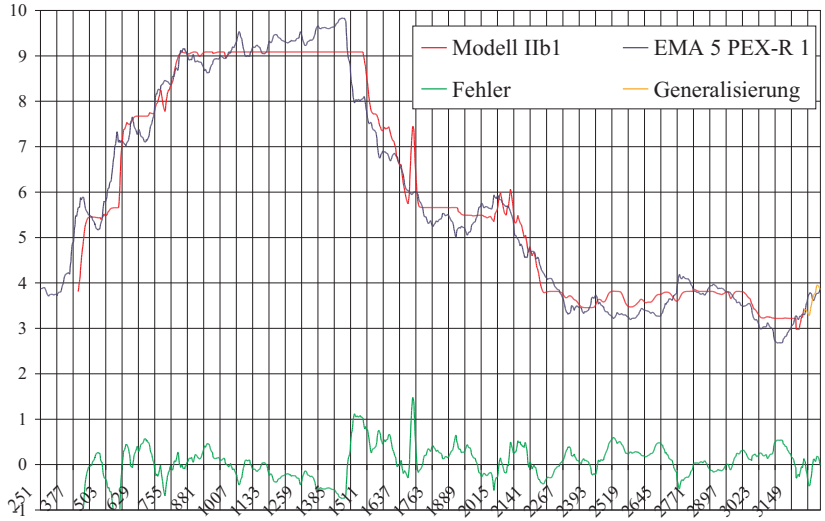


Abbildung 17: Punktprognose von PEX-R 1 auf 3 Monate durch Modell IIb1

Modell IIa3

- PEX-R 1: EMA 10; EMA 21-10, EMA 63-10, EMA 126-10 und Mom_1
  RSI63, TRSI 21/10 und Mom_1
  PEX-R 10: EMA 10, EMA63-10 und Mom_1

n_h	14	A.T.M.	2254	Anteil V.M.	20,10%
F	1,25	A.V.M.	567	Aufteilung	225/63/...

n_h	1		2		3		4		5	
s.c.	n	j	n	j	n	j	n	j	n	j
A.G.	17	31	33	47	49	63	65	79	81	95
A.I.N.	543	13598	2218	21526	4047	25831	6901	31026	10414	35315
A.T.N.	100	100	200	200	300	300	400	400	500	500
Verh.	5,38	134,63	11,09	107,63	13,40	86,10	17,21	77,57	20,75	70,63
$\emptyset\, t_{e.t.N.}$	1,78	44,21	5,47	58,85	5,79	48,76	9,43	48,59	14,36	56,17
ε_t^*	71,18	911,80	59,93	109,78	53,44	31,00	39,38	88,96	54,91	106,00
ε_v^*	70,83	959,23	55,47	124,01	55,80	35,70	40,17	91,89	53,82	85,76
$\emptyset\, \varepsilon_t^*$	0,251	0,899	0,231	0,312	0,218	0,166	0,187	0,281	0,221	0,307
$\emptyset\, \varepsilon_v^*$	0,251	0,251	0,899	0,231	0,312	0,218	0,166	0,187	0,281	0,221

Tabelle 13: Trainingsergebnisse der Prognose von PEX-R 1 auf 3 Monate

	Stützbereich	Generalisierung
Modell	IIa3	
R.T.Q.	76,86%	100,00%
M.A.F.	0,242	0,303
M.Q.F.	0,121	0,102
r	0,989	0,946
R^2	0,978	0,894
U_d	0,512	0,624
U_p	0,619	0,678

Tabelle 14: dazugehörige Prognosegüten

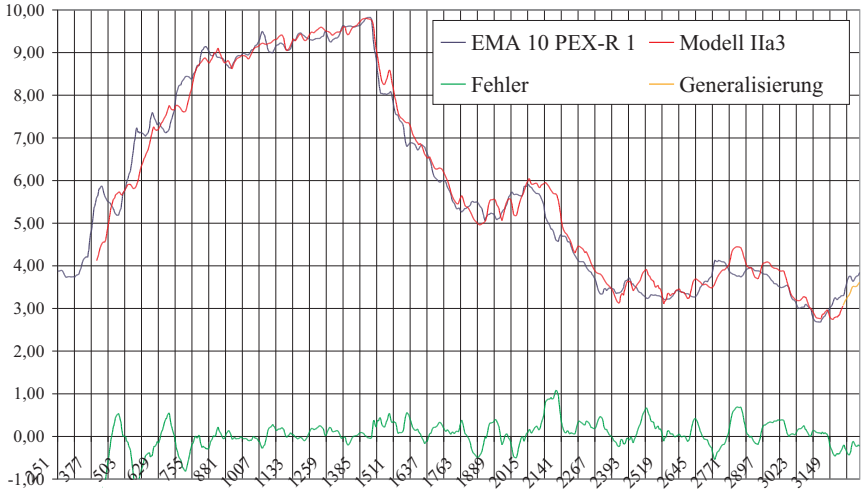


Abbildung 18: Prognose von PEX-R 1 auf 3 Monate durch Modell IIa3

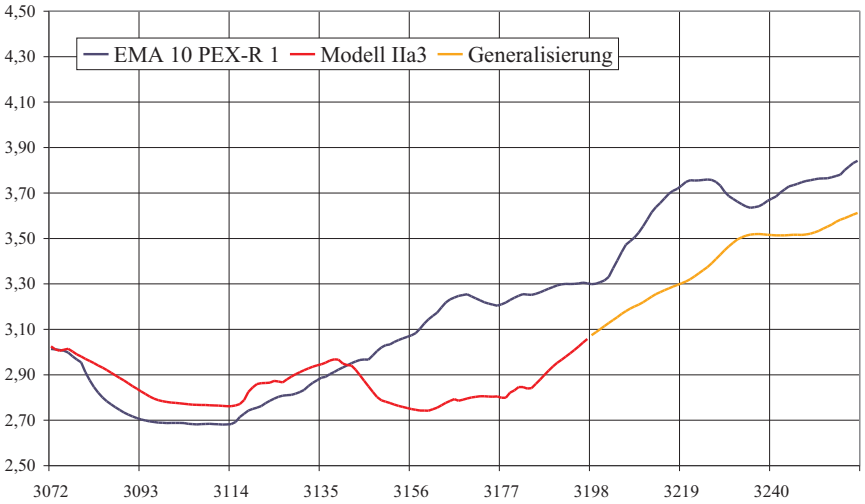


Abbildung 19: Prognose von PEX-R 1 auf 3 Monate durch Modell IIa3 in vergrößerter Darstellung (entspricht 26.3. bis 22.12.1999)

Modell IIa4

- PEX-R 1: EMA 10; EMA 21-10, EMA 42-10, EMA 63-10 und Signaltransformation, RSI 63, TRSI 10/21 Signaltransformation.
  PEX-R 10: EMA 10; EMA42-10 und Signaltransformation

n_h	12	A.T.M.	2254	Anteil V.M.	20,10%
F	1,10	A.V.M.	567	Aufteilung	225/63/...

n_h	1		2		3		4		5	
s.c.	n	j	n	j	n	j	n	j	n	j
A.G.	15	27	29	41	43	55	57	69	71	83
A.I.N.	191	4010	935	6792	1528	9087	2495	12319	4049	23994
A.T.N.	101	97	200	201	300	300	402	400	501	501
Verh.	1,89	41,34	4,68	33,79	5,09	30,29	6,21	30,80	8,08	47,89
$\emptyset\, t_{e.t.N.}$	0,54	11,19	1,71	12,96	2,24	15,33	3,42	16,62	5,19	31,22
ε_t^*	74,15	57,07	55,32	42,11	46,11	36,12	36,38	31,77	32,21	29,13
ε_v^*	75,71	63,77	55,84	38,70	46,62	31,52	38,12	36,18	33,49	28,18
$\emptyset\, \varepsilon_t^*$	0,257	0,225	0,222	0,193	0,202	0,179	0,180	0,168	0,169	0,161
$\emptyset\, \varepsilon_v^*$	0,259	0,238	0,223	0,185	0,203	0,167	0,184	0,179	0,172	0,158

Tabelle 15: Trainingsergebnisse der Prognose von PEX-R 1 auf 3 Monate

	Stützzeitraum	Generalisierung
Modell	IIa4	
R.T.Q.	82,60%	100,00%
M.A.F.	0,220	0,252
M.Q.F.	0,090	0,079
r	0,992	0,852
R^2	0,983	0,726
U_d	0,445	0,548
U_p	0,535	0,596

Tabelle 16: dazugehörige Prognosegüten

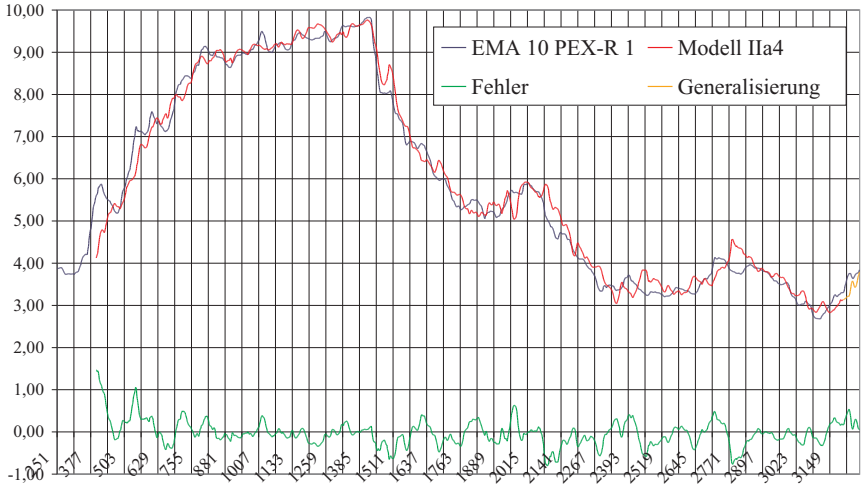


Abbildung 20: Prognose von PEX-R 1 auf 3 Monate durch Modell IIa4

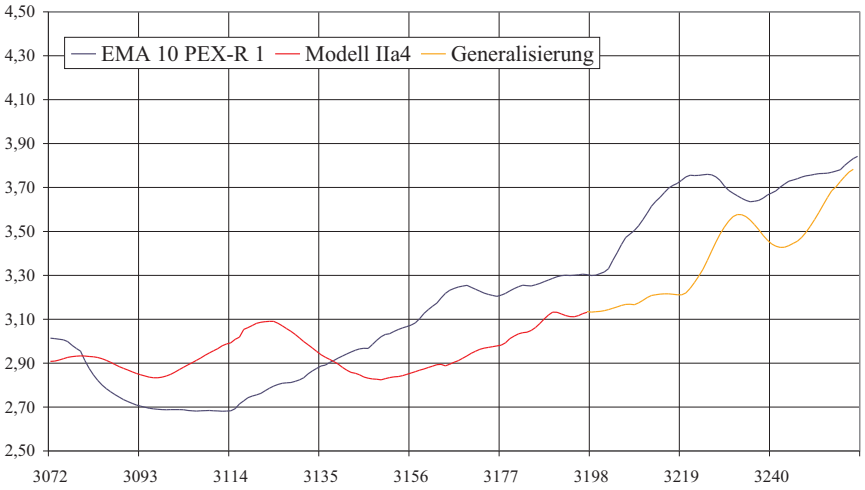


Abbildung 21: Prognose von PEX-R 1 auf 3 Monate durch Modell IIa4 in vergrößerter Darstellung (entspricht 26.3. bis 22.12.1999)

4.2.3 Gewichtsanalyse

Die Anwendung Neuronaler Netze wird zu den *Black-Box* Methoden gerechnet, da nicht explizit nachvollzogen werden kann, *wie* das Netz die ihm gestellten Aufgaben genau löst. Zwar kann die Funktion, welche die Eingabegrößen auf die Ausgaben abbildet, durch die ermittelten Gewichte leicht angegeben und angewandt werden, aber *warum* ausgerechnet dieser Punkt im Gewichtsraum und nicht ein anderer den Trainingsfehler minimiert, bleibt offen. Da diese Situation recht unbefriedigend ist, wurde stets versucht dennoch einen Einblick in die Funktionsweise eines Neuronalen Netzes zu bekommen. Von besonderem Interesse ist hier der Einfluß der Eingabegrößen auf die Ausgabe des Netzes, der theoretisch durch die partiellen Ableitungen der Eingaben nach der Ausgabe gegeben ist. Dadurch wird die Frage beantwortet wie stark die Ausgabe schwankt, wenn eine Eingabegröße verändert wird. Diese Änderung ist aber wiederum abhängig von den Werten der übrigen Eingabegrößen, also dem Punkt im Eingaberaum, an dem die partielle Ableitung betrachtet wird. In der Praxis ist es daher schwierig mit dieser Methode zufriedenstellende Antworten zu erhalten, da die partiellen Ableitungen an vielen Punkten berechnet und bewertet werden müßten, wobei die Frage offen bleibt, an welchen und wievielen Punkten dies genau geschehen soll.

Ein zweiter, eher praktisch orientierter Ansatz, besteht in der der Betrachtung der Gewichte von der Eingabe- zur verdeckten Schicht[7] in der Annahme, daß einflußreichere Eingabegrößen durch höhere Gewichte gekennzeichnet sind und weniger wichtige durch niedrigere Gewichte. Die weiteren Verknüpfungen von der verdeckten zur Ausgabeschicht werden nicht weiter berücksichtigt, da die hier entstehenden Nichtlinearitäten kaum noch auf den Einfluß der Eingabe schließen lassen. Da die Gewichte sowohl negative wie positive Werte annehmen können, wurde zunächst die Betragssumme der Verbindungsgewichte, die von einem Eingabeneuron ausgehen, gebildet. Diese wurde dann zur gesamten Betragssumme der Gewichte von der Eingabe- zur verdeckten Schicht in Beziehung gesetzt, wobei das Bias-Neuron unberücksichtigt blieb. Damit ergibt sich für den Gewichtsanteil G eines Eingabeneurons o_j:

$$G(o_j) = \frac{\sum_{k=1}^{n_h} |w_{j,n_i+k}|}{\sum_{j=1}^{n_i} \sum_{k=1}^{n_h} |w_{j,n_i+k}|} \tag{33}$$

Die Aussagekraft dieses Wertes sollte allerdings nicht überschätzt werden, da schon ähnlich gute Netze einer Topologie völlig unterschiedliche Gewichte haben können. Dies entspricht zwei unterschiedlichen Punkten im Gewichtsraum, denen ähnlich niedrige Trainingsfehler zugeordnet sind. Zusätzlich kommen noch die Unterschiede zwischen den ver-

[7] Bei Netzen mit mehr als drei Lagen entsprechend die erste verdeckte Schicht.

Modell Ia3			Modell IIa1		
Eingabe	PEX-R	Anteil	Eingabe	PEX-R	Anteil
Differenz vor 2 Mon.	1, 10	17,68%	EMA21-5	1	13,89%
Differenz aktuell	1, 10	15,34%	EMA21-5	10	13,79%
Differenz vor 3 Mon.	1, 10	12,49%	EMA63-5	10	11,11%
Differenz akt. - 1 Mon.	1	11,37%	EMA63-5	1	10,32%
Differenz akt. - 2 Mon.	1	10,56%	EMA5	1	9,31%
Differenz akt. - 3 Mon.	1	8,78%	EMA5	10	8,88%
Differenz vor 1 Mon.	1, 10	7,29%	EMA10-5	1	7,70%
Differenz akt. - 2 Mon.	10	7,12%	EMA10-5	10	7,42%
Differenz akt. - 3 Mon.	10	5,72%	RSI63	10	5,83%
Differenz akt. - 1 Mon.	10	3,65%	TRSI10/21	10	4,43%
-		-	TRSI10/21	1	3,72%
-		-	RSI63	1	3,60%

Tabelle 17: Bewertung des Einflusses der Eingabegrößen verschiedener Prognosemodelle durch den relativen Anteil der Betragssumme der Verbindungsgewichte von der Eingabe- zur verdeckten Schicht.

wendeten Topologien zum Tragen, die ebenfalls keine einheitlichen Bewertungen der Eingabegrößen zeigen. Die hier angegebenen Gewichtsanteile entstanden als Mittelwert der in den Modellen aus 4.2.2 verwendeten Topologien und sind lediglich als *Hinweis* auf die Bedeutung einer Eingabegröße zu verstehen. Darüberhinaus ist zu beachten, daß die ermittelten Werte zuverlässige Vergleiche nur innerhalb eines Modells zulassen. Wie stark eine Eingabegröße relativ zu Größen aus anderen Modellen gewichtet worden wäre, kann nur durch Training eines entsprechenden Netzes beantwortet werden.

Im folgenden sollen die Bewertungen der Eingabegrößen der bisher vorgestellten Deltaprognosemodelle von PEX-R 1 auf drei Monate kurz betrachtet werden. In Tabelle 17 sind die Ergebnisse für die Modelle Ia3 und IIa1 der Größe nach sortiert angegeben.Beim Differenzmodell Ia3 fällt der starke Einfluß der Zinsstruktur auf, da drei der vier Differenzen zwischen den Laufzeiten von einem und zehn Jahren an der Spitze liegen. In den folgenden Modellen wurden daher stets auch Eingabegrößen des jeweils entgegengesetzten Laufzeitendes berücksichtigt.[8] Als nächste Gruppe folgen die Differenzen der prognostizierten einjährigen Laufzeit, die erstaunlicherweise nicht die höchste Gewichtung erfuhr. Die Entwicklung von PEX-R 10, repräsentiert durch die gebildeten rückwärtigen Differenzen, ist dagegen erwartungsgemäß relativ niedrig bewertet worden.

Beim Indikatorenmodell IIa1 zeigt sich bezüglich der Laufzeiten ein inhomogeneres Bild ab, während die Dominanz der Gleitenden Durchschnitte und vor allem ihrer Differenzen

[8] Bei der Prognose von PEX-R 1 und 10 wurden zur Einbeziehung der Zinsstruktur jeweils gegenseitig Werte beider Laufzeiten verwendet, beim PEX-R 4 zusätzlich die Werte der zehnjährigen Laufzeit.

Modell IIa2			Modell IIa3		
Eingabe	PEX-R	Anteil	Eingabe	PEX-R	Anteil
EMA63-10	1	10,72%	EMA126-10	1	19,24%
EMA42-10	1	10,01%	EMA63-10	1	14,94%
EMA10	10	9,57%	EMA10	1	10,06%
Mom_1 EMA 10-42	1	8,22%	Mom_1 EMA21-10	1	9,59%
EMA42-10	10	8,18%	Mom_1 EMA63-10	1	9,40%
EMA21-10	1	7,44%	Mom_1 EMA126-10	1	7,29%
Mom_1 TRSI10/21	1	7,34%	EMA21-10	1	6,65%
EMA10	1	7,13%	EMA10	10	5,30%
RSI63	1	6,53%	EMA63-10	10	5,26%
Mom_1 EMA10-21	1	6,02%	RSI63	1	3,53%
TRSI10/21	1	5,67%	TRSI10/21	1	3,47%
Mom_1 EMA63-10	1	4,92%	Mom_1 TRSI10/21	1	2,29%
Mom_1 RSI63	1	4,23%	Mom_1 RSI63	1	2,07%
Mom_1 EMA42-10	10	4,02%	Mom_1 EMA63-10	10	0,92%

Tabelle 18: Bewertung des Einflusses der Eingabegrößen bei den Indikatormodellen mit Berücksichtigung des Eintagesmomentums.

klar zu erkennen ist. Da die Zeitreihe des PEX bzw. seiner Renditen zum überwiegenden Teil durch eindeutige, mindestens mittelfristig anhaltende Trends gekennzeichnet ist, kann dieses Ergebnis nicht überraschen. Trotzdem, oder gerade deshalb, wurden die schwach bewerteten Oszillatoren RSI und TRSI auch in den weiteren Modellen verwendet, um die bisher eher seltenen Seitwärtsbewegungen im Falle ihres Auftretens berücksichtigen zu können. Bei der weiteren Modellentwicklung wurden aufgrund der Ergebnisse beim Modell IIa1 bei der zehnjährigen Laufzeit nur noch der geglättete Basiswert, eine Differenz Gleitender Durchschnitte und eine Transformation verwendet, um sich mehr auf Eingabegrößen der zu prognostizierenden Laufzeit zu konzentrieren.

Die Ergebnisse der Modelle IIa2 und IIa3 (Tabelle 18), bei denen teilweise auch das Eintagesmomentum der Indikatoren als Eingabegröße verwendet wurde, zeigen keine einheitliche Struktur bezüglich der Bewertung der Eingaben. Das Modell IIa2 zeigt nur relativ geringe Unterschiede zwischen den Eingabegrößen, allerdings kann auch hier von einer Bevorzugung der Trendfolger gegenüber den Oszillatoren gesprochen werden. Auffällig bei beiden Modellen ist, daß gerade drei Momentumswerte das Schlußlicht bilden. Dies kann dadurch erklärt werden, daß das Momentum nur in wenigen Situationen ausschlaggebend für eine Prognose ist, z.B. beim Einsetzen einer Trendumkehr. Ebenso kann eine nicht untersuchte Redundanz unter den Momentumswerten vorliegen, die sich eventuell auch auf die Differenz Gleitender Durchschnitte erstreckt. Da sich die beiden Modelle nur durch die Zeitparameter ihrer Gleitenden Durchschnitte unterscheiden, ist ein Blick auf

Modell IIa4			Modell IIa5		
Eingabe	PEX-R	Anteil	Eingabe	PEX-R	Anteil
EMA42-10	1	30,18%	EMA63-10	1	25,33%
EMA63-10	1	24,43%	EMA126-10	1	24,56%
EMA21-10	1	13,21%	EMA21-10	1	9,95%
EMA10	1	10,45%	EMA10	10	8,56%
EMA10	10	5,85%	EMA10	1	6,33%
Signal RSI63	1	4,50%	Signal EMA126-10	1	5,59%
Signal TRSI	1	3,21%	Signal TRSI10/21	1	4,30%
Signal EMA63-10	1	2,55%	Signal RSI63	1	4,02%
EMA42-10	10	1,92%	EMA63-10	10	3,66%
Signal EMA42-10	10	1,72%	Signal EMA63-10	10	3,46%
Signal EMA42-10	1	1,16%	Signal EMA21-10	1	2,42%
Signal EMA21-10	1	0,81%	Signal EMA63-10	1	1,83%

Tabelle 19: Bewertung des Einflusses der Eingabegrößen bei den Indikatormodellen mit Signaltransformationen.

die entsprechenden Indikatoren von besonderem Interesse, auch wenn ein direkter Vergleich aus den genannten Gründen nicht möglich ist. So werden beim Modell IIa3 mit seinen längerfristigen GDs diese zwar stärker berücksichtigt, sind in der Reihenfolge untereinander aber ähnlich einzustufen. So wird die kurzfristige Differenz EMA21-10 des PEX-R 1 eher schwach eingestuft, während die langfristigeren Differenzen jeweils die beiden Spitzenplätze einnehmen. Ebenso wie bei der Betrachtung der Trainingsergebnisse und der Prognosegüten kann hier keine Entscheidung über die bessere Parameterwahl getroffen werden.

Zu einem analogen Schluß führt die Auswertung der Ergebnisse der Modelle IIa4 und IIa5 (Tabelle 19), die zusätzlich Signaltransformationen einiger Indikatoren als Eingabegrößen haben. Zwar werden hier im Unterschied zu IIa2 und IIa3 gerade die Differenzen der kürzeren GDs stärker bewertet, allerdings ist die Reihenfolge innerhalb eines Modells absolut identisch. Dies gilt bis auf wenige Ausnahmen interessanterweise ebenso für die übrigen Eingabegrößen. Weiterhin ist festzuhalten, daß die Signaltransformationen von RSI und TRSI in diesen Modellen stärkerer Einfluß zugebilligt wird, als dies bei den Momentumswerten der Fall ist. Im Gegenzug liegen dafür die Signaltransformationen dreier Gleitender Durchschnitte auf den letzten Plätzen.

Insgesamt bleibt festzustellen, daß die letzten vier Indikatorenmodelle die untransformierten Trendfolger bevorzugen, während transformierte Größen und Oszillatoren von ihrer Bedeutung her geringer eingestuft werden. Dies sollte aber nicht dazu verleiten, diese scheinbar unwichtigen Eingabegrößen einfach zu eliminieren, in der Hoffnung, das Netz-

training zu vereinfachen und dennoch gleich gute Ergebnisse zu erzielen. Ich selbst habe an einem Testbeispiel (siehe Anhang, S. 150), das hier nicht weiter behandelt werden soll, festgestellt, daß das Weglassen selbst weniger, scheinbar unbedeutender Eingabegrößen die Prognosegüte wesentlich reduzieren kann. Dies beeinträchtigt andererseits nicht die getroffene Aussage, möglichst wenig redundante Eingabegrößen auszuwählen. Bei dem erwähnten Modell wurden ausschließlich Differenzen Gleitender Durchschnitte verwendet, um festzustellen, ob eher die lang- oder kurzfristigen Differenzen ausschlaggebend sind. Genau wie bei den hier vorgestellten Modellen konnte diese Frage nicht eindeutig beantwortet werden, allerdings zeigte sich, daß bei nahe aneinanderliegenden Indikatoren (z.B. EMA21-10 und EMA31-10) einer stark und der andere schwach gewichtet wurden, was als Indiz für redundante Eingaben angesehen werden kann. Nachdem die Eingabegrößen mit den kleinsten Gewichten weggelassen wurden, zeigte sich allerdings eine wesentlich schlechtere Prognosegüte. Als Mahnung und Anregung sei daher ein Zitat von Masters angeführt, daß sich auch in [Poddig, S. 260][9] findet:

> „The first few principal components certainly explain the majority of the variation in the data, but that implies nothing about classification ability. For all we know, the classification ability may reside in the *last* principal component!"

4.3 Endgültiges Modell

Nach Vergleich der statistischen Werte und einer subjektiven visuellen Beurteilung der Prognosekurven wurde schließlich das Modell IIa4 ausgewählt, um ein vollständiges mittelfristiges Prognosesystem für die Renditen des deutschen Pfandbriefindex zu bilden. Das Modell wurde erneut für die drei Prognosehorizonte von drei, sechs und neun Monaten für Laufzeiten von einem, vier und zehn Jahren trainiert.

Zur benutzerfreundlichen Anwendung des gewählten Prognosemodells wurde dieses innerhalb eines Tabellenkalkulationsprogramms[10] implementiert. Das Ergebnis des Trainings Neuronaler Netze mit FAUN bzw. FAUN-PVM ist lediglich eine unanschauliche Liste von Zahlen, eben dem gefundenen Punkt aus dem Gewichtsraum, bei dem der Trainingsfehler unter Beachtung der Übertrainingsbedingung (13) minimiert werden konnte. Diese Liste von Gewichten muß zunächst in eine entsprechende Netzfunktion übersetzt werden. Dies geschieht hier durch ein MAPLE-Skript, daß Bestandteil des FAUN 0.2-Pakets ist und optimierten FORTRAN- und C-Code für trainierte Neuronale Netze liefert. Die

[9] Poddig gibt zu dem Zitat zwar eine Literaturstelle an, diese findet sich im Literaturverzeichnis allerdings nicht. Daher kann ich die genaue Quelle leider nicht angeben.

[10] Microsoft Excel 97, das vor allem durch die Möglichkeiten der grafischen Darstellung bevorzugt wurde.

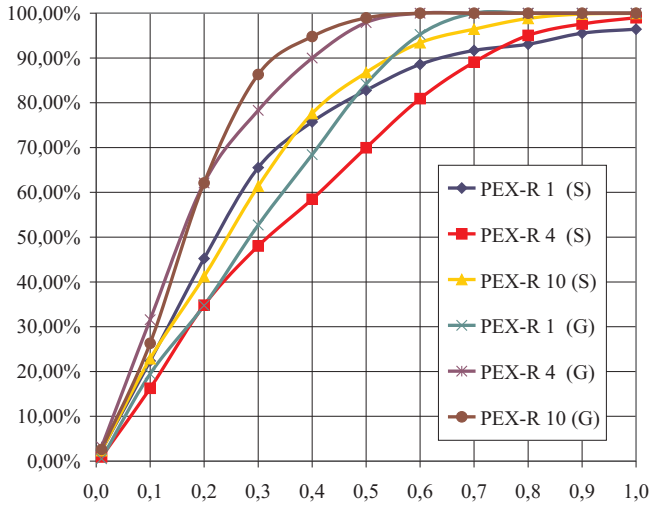


Abbildung 22: Trefferquoten der Prognose von PEX-R 1, 4 und 10 auf 3 Monate abhängig von der zugelassenen Abweichung für den Stütz- (S) und Generalisierungszeitraum (G), aus denen sich die 60% und 80%-Linien für das Gesamtmodell ergeben.

Netzfunktion als Essenz der aufwendigen Datenvorbereitungen und Berechnungen kann schließlich relativ einfach in eine Tabellenkalkulation eingefügt werden.

Nachfolgend werden ähnlich wie bei den bereits vorgestellten Modellen die Daten der Topologie, der Musteraufteilung, der Trainingsergebnisse und die Prognosegüten jeweils für einen Prognosehorizont mit den drei Laufzeiten von 1, 4 und 10 Jahren vorgestellt. Zusätzlich zu dem durchschnittlichen Fehler $\emptyset\varepsilon^*_{t/v} = \sqrt{2 \cdot \frac{\varepsilon^*_{t/v}}{n_{t/v}}}$ [11] pro Muster ist dieser noch „unskaliert" angegeben, d.h. umgerechnet auf die tatsächliche Bandbreite der Werte statt des Intervalls [-0,95; 0,95]. Dadurch ergibt sich eine unverzerrtere, absolute Vergleichsmöglichkeit zwischen den Laufzeiten und Prognosehorizonten. Dennoch ist zu beachten, daß diese groben Überschlagswerte deutlich vom tatsächlichen mittleren absoluten Fehler abweichen können.

Bei der Implementation in die Tabellenkalkulation wurden zwei weitere Kenngrößen hinzugefügt, die eine Beurteilung zukünftiger Prognosegüten erleichtern soll. So sind in der aktuellen (vergrößerten) grafischen Darstellung eine 60% und eine 80%-Linie angegeben. Diese ergeben sich aus dem Prognosewert plus/minus der historischen Abweichung vom tatsächlichen Wert, bei der eine 60 bzw. 80%ige Trefferquote erzielt wurde. Eine typische Trefferquotenverteilung für zugelassene Abweichungen von 0,01% bis zu 1% vom tatsächlichen Wert bei der Prognose auf drei Monate ist in Abb. 22 dargestellt. Im Stützzeitraum zeigen PEX-R 1 und 10 ein ähnliches Verhalten, während der PEX-R 4 insgesamt schlechter abschneidet. Im Generalisierungszeitraum reißt dagegen die einjährige Laufzeit nach unten aus, wogegen die anderen Laufzeiten sogar deutlich besser abschneiden als im Stützzeitraum. Dies deckt sich mit den Aussagen, die sich aus den anderen Gütemaßstäben aus Tabelle 23 auf S. 79 ergeben.

[11] vgl. (11), (12) auf S. 12

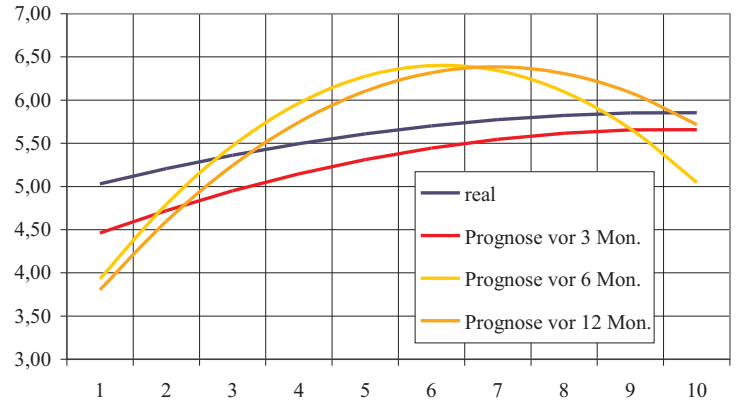


Abbildung 23: Darstellung der aktuellen, realen Zinsstruktur (7.7.2000) und der interpolierten Prognosen vor 3, 6 und 12 Monaten

Weiterhin sind zwei „Perioden"-Linien eingezeichnet, die sich durch Addition bzw. Subtraktion des historischen Fehlers ergeben, der im Zeitrahmen des Prognosehorizontes zurückliegend aufgetreten ist. Dieser zusätzliche Indikator entstand durch die Beobachtung, daß gute Treffer fast periodisch in Abständen des Prognosehorizontes erzielt wurden, wie an den Nullstellen der Fehlerlinien gut abzulesen ist. Treffen sich die Prognose- und die Periodenlinien, wird von einer erhöhten Wahrscheinlichkeit für einen guten Treffer ausgegangen. Inwieweit sich diese Annahme in der Zukunft bestätigt, d.h. diese quasi-Periodizität erhalten bleibt, ist abzuwarten. Bei den Prognosen von PEX-R 4 und 10 auf drei Monate hat sich dieses System bisher schon gut bewährt, wie den Abb. 27 und 29 zu entnehmen ist. Bei der unvermeidlichen Interpretation[12] der Prognosergebnisse sollte eine weitere Beobachtung einfließen, die sich direkt aus der Quasiperiodizität des Fehlers ergibt. Das Hin- und Herschwingen der Abweichung entspricht einer zu niedrigen Prognose bei Aufwärts- und einer zu hohen bei Abwärtstrends. Generell finden Prognose und realer Wert aber immer wieder zusammen, so daß von einer Art „Anziehungskraft" der beiden

[12] Eine Prognose – gleich welcher Herkunft – sollte immer im Zusammenhang mit anderen Quellen und den eigenen Erfahrungen kritisch hinterfragt und nicht als fest gegeben angesehen werden.

Werte zu einem (instabilen) Gleichgewicht gesprochen werden kann, die an dynamische Systeme erinnert. Die momentane Abweichung des Wertes der zurückliegenden Prognose vom aktuellen realen Wert kann somit als Relativierung der Prognose am Ende des betrachteten Horizonts verwendet werden, da auf lange Sicht mit einem Schneiden der Linien und einem „Seitenwechsel" zu rechnen ist.

Bei der Interpolation der restlichen Laufzeiten zur Abbildung der Zinsstruktur wurde auf eine dreidimensionale Darstellung wie in Abb. 2 auf S. 3 verzichtet. Diese ist in der Anwendung eher unpraktisch, da realer Wert und verschiedene Prognosen nicht zusammen darstellbar sind. Stattdessen wurde eine zweidimensionale Darstellung gewählt, wie sie in Abb. 23 zu sehen ist. Hier wurde der aktuelle reale Wert (7.7.2000) den Prognosen von vor drei, sechs und zwölf Monaten gegenübergestellt. Hier ist zu erkennen, daß der kürzeste Prognosehorizont der Zinsstruktur am nächsten kommt, während die weiter zurückliegenden Prognosen geringere Werte im einjährigen und höhere im vierjährigen Bereich erwartet haben.

Dieser Unterschied ist besonders mit Blick auf die einzelnen Prognoselinien interessant, da diese vor einem halben und einem Jahr noch von sinkenden Zinssätzen im einjährigen Bereich ausgegangen sind. Durch die Leitzinserhöhungen der Europäischen Zentralbank im Frühjahr 2000 sind die Zinsen aber weiter gestiegen, so daß sich alle Modelle beim PEX-R 1 momentan kräftig nach oben korrigieren, und eine stark naive Prognose für alle Horizonte liefern. Überhaupt ist allen betrachteten Modellen dieses „naive Moment" gemeinsam, auf dessen Dämpfung sich weitere Untersuchungen konzentrieren sollten. Sehr interessant ist auch die Tatsache, daß alle Modelle vergleichbare gute und schlechte Prognosephasen haben, völlig unabhängig von den konkret verwendeten Parametern. Dafür scheinen modellunabhängige Einflüsse, z.B. untypische Wirtschafts- und Zinsentwicklungen in den schlechten Prognosephasen, verantwortlich zu sein. Insgesamt ergab sich für den Stützzeitraum, daß die vierjährige Laufzeit am schwierigsten, die zehnjährige am einfachsten zu prognostizieren war. Letzteres kann vermutlich auf den regelmäßigeren Verlauf des PEX-R 10 zurückgeführt werden. Auch im Generalisierungszeitraum kann die zehnjährige Laufzeit überzeugen, während beim PEX-R 4 zufriedenstellende Ergebnisse erzielt werden. Die einjährige Laufzeit zeigt dagegen schlechtere Prognosergebnisse, da hier am „kurzen Ende" der Zentralbankeinfluß, der hier besonders groß ist, offensichtlich nicht vorhergesehen werden konnte. Hier bleibt abzuwarten, inwieweit sich Prognose und realer Verlauf zueinander entwickeln, sobald ökonomisch „normale" oder zumindest erlernte Muster wieder auftreten.

Die Trainingsergebnisse zeigen, daß das ausgewählte Modell eindeutig besser mit Shortcuts funktioniert, so daß hier eine einheitliche XPC-Topologie der entsprechenden fünf Netze pro Horizont und Laufzeit gebildet wurde. Dieses eindeutige Ergebnis zugunsten

von Shortcuts ist nicht als selbstverständlich anzusehen und sollte nicht verallgemeinert werden. Bei den Modellen, die zur Auswahl standen, waren ebenso zahlreiche Topologien zu finden, die ohne Shortcuts besser abschnitten (siehe 4.2.1 und4.2.2). Die etwas ungewöhnliche Aufteilung von Trainings- und Validierungsmustern ergab sich aus dem Bemühen, saisonale Einflüsse gleichmäßig zu berücksichtigen. Da ein Jahr aus ca. 250 Handelstagen besteht, würden bei einer ähnlich strukturierten Aufteilung immer die gleichen Jahreszeiten in den einzelnen Mengen zu finden sein, was offensichtlich vermieden werden sollte. Der Vergleich der Prognose von PEX-R 1 auf drei Monate mit den Ergebnissen aus der Modellauswahlphase, bei der im wesentlichen andere Trainings- und Validierungsmengen verwendet wurden, unterstreicht die beschriebene Abhängigkeit des Trainings von der Datenaufteilung.

4.3.1 Prognose auf 3 Monate

n_i	12	A.T.M.	2263	Anteil	21,64%
F	1,20	A.V.M.	625	Aufteilung T/V/T/V/...	500/125/250/125/...

n_h	1		2		3		4		5	
s.c.	n	j	n	j	n	j	n	j	n	j
A.G.	15	27	29	41	43	55	57	69	71	83
A.I.N.	102	103	202	216	321	317	424	451	539	605
A.T.N.	100	100	200	200	300	300	400	400	500	500
Verh.	1,02	1,03	1,01	1,08	1,07	1,06	1,06	1,13	1,08	1,21
$\emptyset\, t_{e.t.N.}$	0,58	0,35	0,28	0,46	0,53	0,43	0,35	0,81	0,56	1,04
ε_t^*	78,93	62,18	64,09	54,80	54,91	46,89	53,67	39,59	48,06	42,26
ε_v^*	72,54	38,37	64,67	37,47	62,16	36,71	44,77	41,83	31,18	40,36
$\emptyset\, \varepsilon_t^*$	0,264	0,234	0,238	0,220	0,220	0,204	0,218	0,187	0,206	0,193
$\emptyset\, \varepsilon_v^*$	0,253	0,184	0,239	0,182	0,234	0,180	0,199	0,192	0,166	0,189
$\emptyset\, \varepsilon_t^*$unsk.	0,524	0,465	0,472	0,437	0,437	0,404	0,432	0,371	0,409	0,383
$\emptyset\, \varepsilon_v^*$unsk.	0,502	0,365	0,474	0,361	0,465	0,357	0,395	0,381	0,329	0,375

Tabelle 20: Trainingsergebnisse der Prognose von PEX-R 1 auf 3 Monate

n_i	12	A.T.M.	2263	Anteil	21,64%
F	1,20	A.V.M.	625	Aufteilung	500/125/250/125/...

n_h	1		2		3		4		5	
s.c.	n	j	n	j	n	j	n	j	n	j
A.G.	15	27	29	41	43	55	57	69	71	83
A.I.N.	110	113	211	265	344	456	473	731	624	1119
A.T.N.	100	100	200	200	300	300	400	400	500	500
Verh.	1,10	1,13	1,06	1,33	1,15	1,52	1,18	1,83	1,25	2,24
$\emptyset\, t_{e.t.N.}$	0,23	0,34	0,26	0,74	0,33	0,80	0,60	1,06	0,79	1,48
ε_t^*	121,8	99,4	105,2	92,2	100,2	76,7	91,1	80,0	89,0	71,0
ε_v^*	132,9	109,9	122,3	102,3	112,2	91,0	101,1	90,0	99,5	78,3
$\emptyset\, \varepsilon_t^*$	0,328	0,296	0,305	0,285	0,298	0,260	0,284	0,266	0,280	0,250
$\emptyset\, \varepsilon_v^*$	0,343	0,312	0,329	0,301	0,315	0,284	0,299	0,282	0,297	0,263
$\emptyset\, \varepsilon_t^*$unsk.	0,474	0,428	0,440	0,412	0,430	0,376	0,410	0,384	0,405	0,362
$\emptyset\, \varepsilon_v^*$unsk.	0,495	0,450	0,475	0,434	0,455	0,410	0,432	0,407	0,428	0,380

Tabelle 21:Trainingsergebnisse der Prognose von PEX-R 4 auf 3 Monate

n_i	12	A.T.M.	2263	Anteil	21,64%
F	1,20	A.V.M.	625	Aufteilung	500/125/250/125/...

n_h	1		2		3		4		5	
s.c.	n	j	n	j	n	j	n	j	n	j
A.G.	15	27	29	41	43	55	57	69	71	83
A.I.N.	108	221	240	495	392	682	545	863	730	1078
A.T.N.	100	100	200	200	300	300	400	400	500	500
Verh.	1,08	2,21	1,20	2,48	1,31	2,27	1,36	2,16	1,46	2,16
$\emptyset\, t_{e.t.N.}$	0,28	0,38	0,27	0,49	0,28	0,55	0,34	0,54	0,44	0,67
ε_t^*	120,0	101,8	103,3	96,0	102,0	104,7	93,4	96,2	85,5	82,6
ε_v^*	116,5	105,8	108,3	107,9	104,3	119,4	111,8	114,9	97,6	98,8
$\emptyset\, \varepsilon_t^*$	0,326	0,300	0,302	0,291	0,300	0,304	0,287	0,292	0,275	0,270
$\emptyset\, \varepsilon_v^*$	0,321	0,306	0,309	0,309	0,304	0,325	0,314	0,319	0,294	0,296
$\emptyset\, \varepsilon_t^*$unsk.	0,329	0,303	0,305	0,294	0,303	0,307	0,290	0,295	0,278	0,273
$\emptyset\, \varepsilon_v^*$unsk.	0,324	0,309	0,313	0,312	0,307	0,328	0,318	0,322	0,297	0,299

Tabelle 22: Trainingsergebnisse der Prognose von PEX-R 10 auf 3 Monate

	Stützbereich			Generalisierung		
PEX-R	1	4	10	1	4	10
R.T.Q.	70,90%	67,30%	71,40%	97,90%	92,10%	69,50%
60%	0,25	0,37	0,27	0,37	0,19	0,19
80%	0,42	0,55	0,39	0,46	0,33	0,27
M.A.F.	0,286	0,34	0,254	0,262	0,169	0,184
M.Q.F.	0,159	0,173	0,097	0,094	0,045	0,044
r	0,983	0,968	0,966	0,898	0,711	0,188
R^2	0,966	0,937	0,933	0,806	0,506	0,035
U_d	0,569	0,619	0,659	1,148	0,475	0,394
U_p	0,703	0,761	0,777	0,556	0,530	0,696

Tabelle 23: Prognosegüten PEX-R 1, 4 und 10 auf 3 Monate

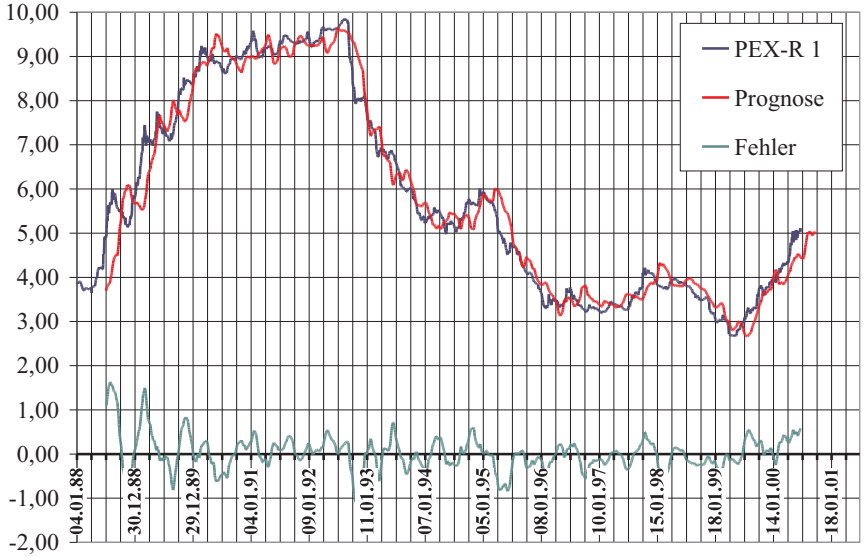


Abbildung 24: Prognose PEX-R 1 auf 3 Monate

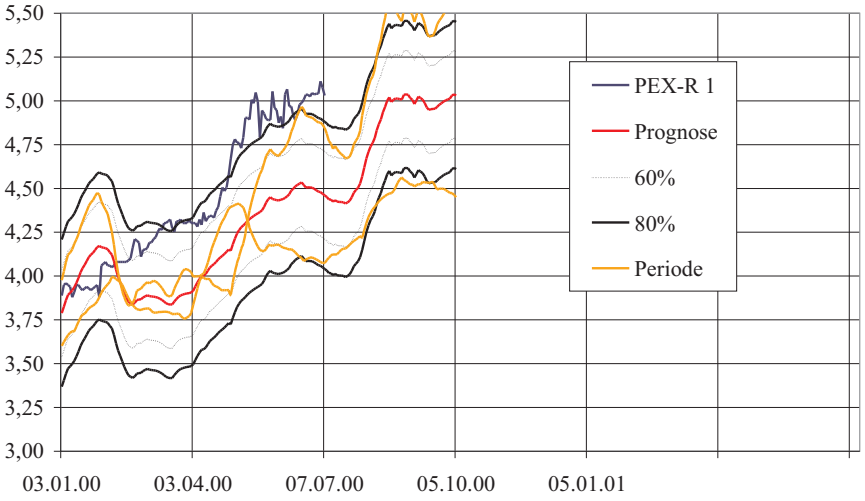


Abbildung 25: Prognose PEX-R 1 auf 3 Monate (vergrößert)

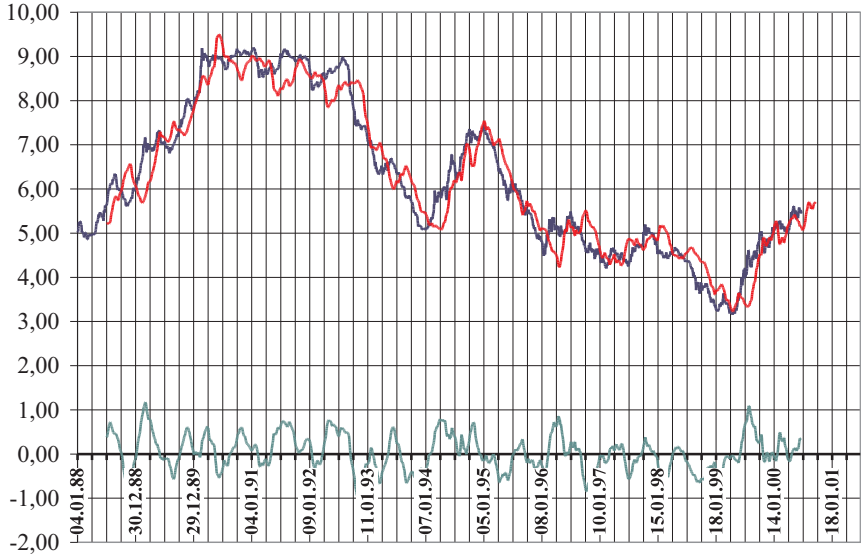


Abbildung 26: Prognose PEX-R 4 auf 3 Monate

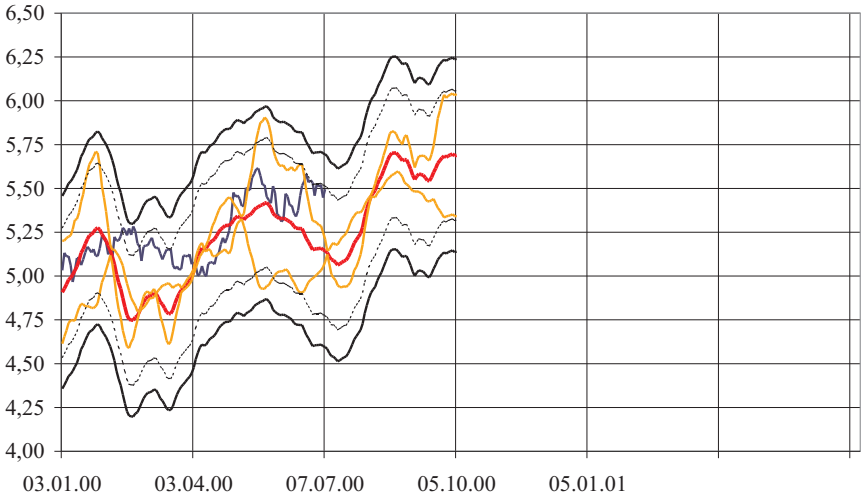


Abbildung 27: Prognose PEX-R 4 auf 3 Monate (vergrößert)

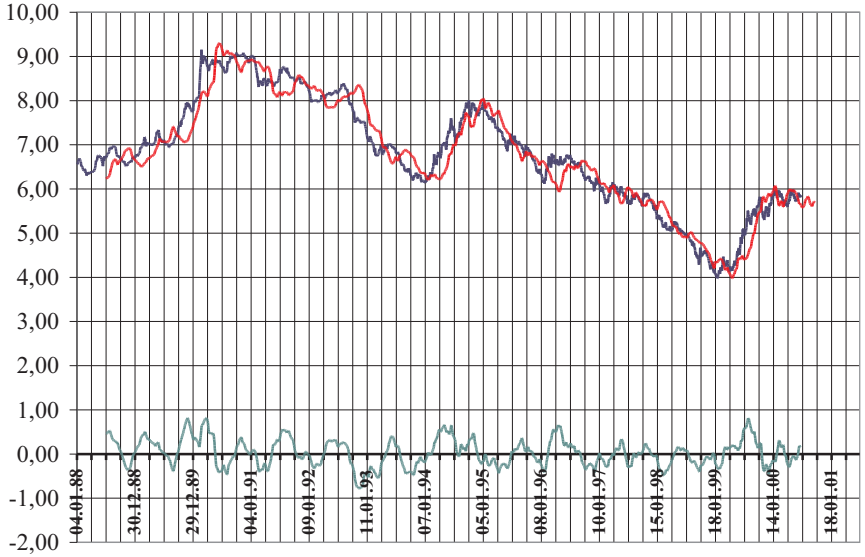


Abbildung 28: Prognose PEX-R 10 auf 3 Monate

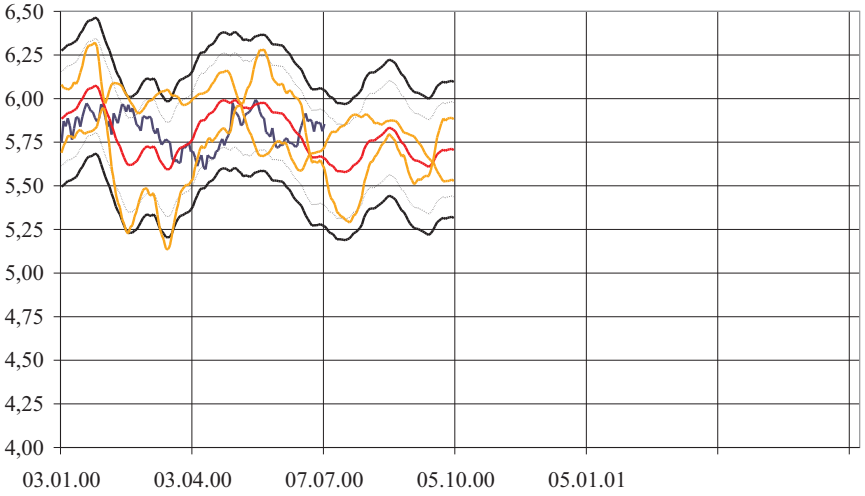


Abbildung 29: Prognose PEX-R 10 auf 3 Monate (vergrößert)

4.3.2 Prognose auf 6 Monate

n_i	12	A.T.M.	2209	Anteil	19,93%
F	1,20	A.V.M.	550	Aufteilung	490/110/245/110/...

n_h	1		2		3		4		5	
s.c.	n	j	n	j	n	j	n	j	n	j
A.G.	15	27	29	41	43	55	57	69	71	83
A.I.N.	132	516	280	1278	451	1669	654	2539	797	3489
A.T.N.	100	100	200	200	300	300	400	400	500	500
Verh.	1,32	5,16	1,40	6,39	1,50	5,56	1,64	6,35	1,59	6,98
$\emptyset\, t_{e.t.N.}$	0,25	0,84	0,26	1,29	0,29	1,45	0,35	1,87	0,42	2,39
ε_t^*	70,33	42,60	31,96	35,49	44,18	24,03	45,85	18,76	32,65	18,67
ε_v^*	52,28	49,29	31,36	41,16	41,91	25,52	48,65	22,05	36,95	19,90
$\emptyset\, \varepsilon_t^*$	0,252	0,196	0,170	0,179	0,200	0,148	0,204	0,130	0,172	0,130
$\emptyset\, \varepsilon_v^*$	0,218	0,211	0,169	0,193	0,195	0,152	0,210	0,141	0,183	0,134
$\emptyset\, \varepsilon_t^*$unsk.	0,696	0,541	0,469	0,494	0,551	0,407	0,562	0,359	0,474	0,358
$\emptyset\, \varepsilon_v^*$unsk.	0,600	0,582	0,464	0,532	0,537	0,419	0,579	0,389	0,504	0,370

Tabelle 24: Trainingsergebnisse der Prognose von PEX-R 1 auf 6 Monate

n_i	12	A.T.M.	2209	Anteil	19,93%
F	1,20	A.V.M.	550	Aufteilung	490/110/245/110/...

n_h	1		2		3		4		5	
s.c.	n	j	n	j	n	j	n	j	n	j
A.G.	15	27	29	41	43	55	57	69	71	83
A.I.N.	130	277	261	1006	444	1749	609	2847	856	
A.T.N.	100	100	200	200	300	300	400	400	500	500
Verh.	1,30	2,77	1,31	5,03	1,48	5,83	1,52	7,12	1,71	0,00
$\emptyset\, t_{e.t.N.}$	0,27	0,53	0,30	1,02	0,33	1,50	0,48	2,28	0,65	3,01
ε_t^*	118,7	97,2	92,7	81,3	94,5	60,0	84,0	68,3	80,4	60,7
ε_v^*	124,8	113,8	102,7	74,0	88,1	67,9	68,4	80,6	89,6	72,5
$\emptyset\, \varepsilon_t^*$	0,328	0,297	0,290	0,271	0,292	0,233	0,276	0,249	0,270	0,234
$\emptyset\, \varepsilon_v^*$	0,336	0,321	0,305	0,259	0,282	0,248	0,249	0,270	0,285	0,256
$\emptyset\, \varepsilon_t^*$unsk.	0,738	0,667	0,652	0,610	0,658	0,525	0,620	0,560	0,607	0,527
$\emptyset\, \varepsilon_v^*$unsk.	0,756	0,722	0,686	0,582	0,635	0,558	0,560	0,608	0,641	0,576

Tabelle 25: Trainingsergebnisse der Prognose von PEX-R 4 auf 6 Monate

n_i	12	A.T.M.	2209	Anteil	19,93%
F	1,20	A.V.M.	550	Aufteilung	490/110/245/110/...

n_h	1		2		3		4		5	
s.c.	n	j	n	j	n	j	n	j	n	j
A.G.	15	27	29	41	43	55	57	69	71	83
A.I.N.	113	441	248	670	393	932	518	1270	721	1387
A.T.N.	100	100	200	200	300	300	400	400	500	500
Verh.	1,13	4,41	1,24	3,35	1,31	3,11	1,30	3,18	1,44	2,77
$\emptyset t_{e.t.N.}$	0,24	0,70	0,26	0,73	0,28	0,81	0,33	0,91	0,45	0,95
ε_t^*	89,50	74,26	54,13	32,36	63,48	27,77	52,44	27,64	54,36	17,72
ε_v^*	55,33	72,11	60,61	33,47	62,85	32,19	59,80	31,19	50,42	21,23
$\emptyset \varepsilon_t^*$	0,285	0,259	0,221	0,171	0,240	0,159	0,218	0,158	0,222	0,127
$\emptyset \varepsilon_v^*$	0,224	0,256	0,234	0,174	0,239	0,171	0,233	0,168	0,214	0,139
$\emptyset \varepsilon_t^*$unsk.	0,493	0,449	0,383	0,296	0,415	0,275	0,377	0,274	0,384	0,219
$\emptyset \varepsilon_v^*$unsk.	0,388	0,443	0,406	0,301	0,413	0,296	0,403	0,291	0,370	0,240

Tabelle 26: Trainingsergebnisse der Prognose von PEX-R 10 auf 6 Monate

	Stützbereich			Generalisierung		
PEX-R	1	4	10	1	4	10
R.T.Q.	80,40%	76,90%	88,20%	85,21%	100,00%	82,10%
60%	0,310	0,460	0,210	0,430	0,500	0,460
80%	0,47	0,740	0,320	0,750	0,740	0,640
M.A.F.	0,301	0,451	0,198	0,378	0,477	0,332
M.Q.F.	0,143	0,316	0,062	0,250	0,308	0,180
r	0,986	0,946	0,979	0,814	0,845	0,743
R^2	0,972	0,895	0,959	0,662	0,714	0,552
U_d	0,443	0,622	0,384	0,636	0,554	0,467
U_p	0,441	0,670	0,404	0,919	1,009	1,039

Tabelle 27: Prognosegüten PEX-R 1, 4 und 10 auf 6 Monate

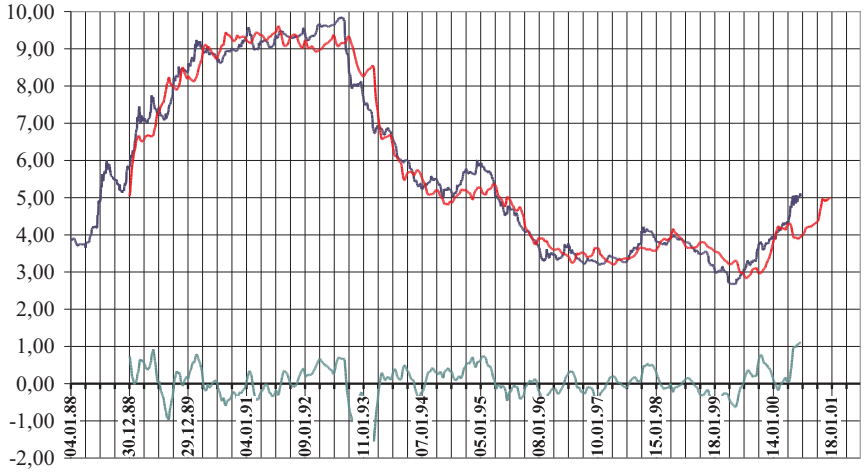


Abbildung 30: Prognose PEX-R 1 auf 6 Monate

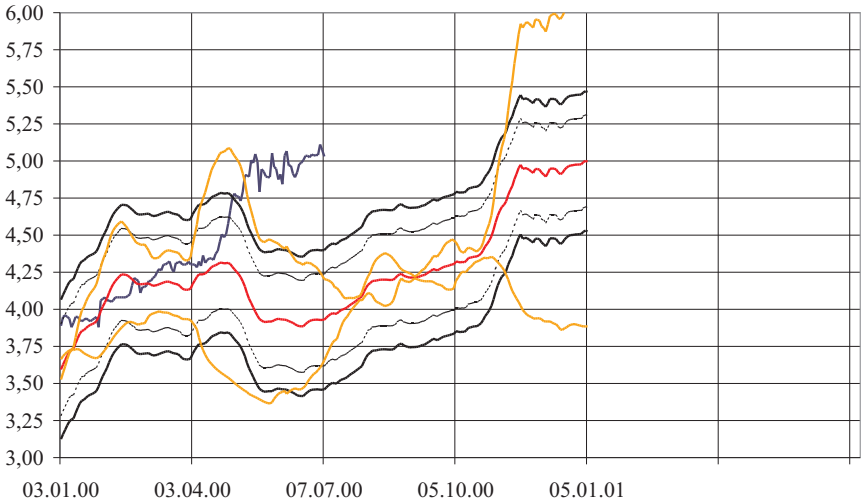


Abbildung 31: Prognose PEX-R 1 auf 6 Monate (vergrößert)

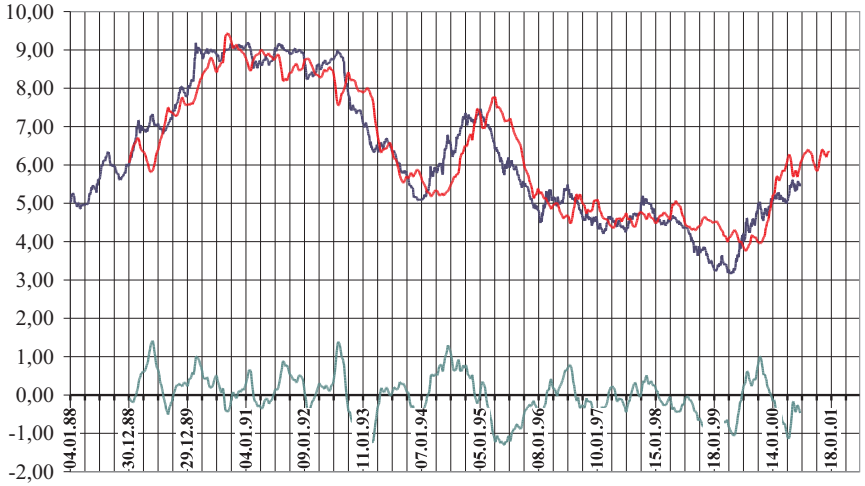


Abbildung 32: Prognose PEX-R 4 auf 6 Monate

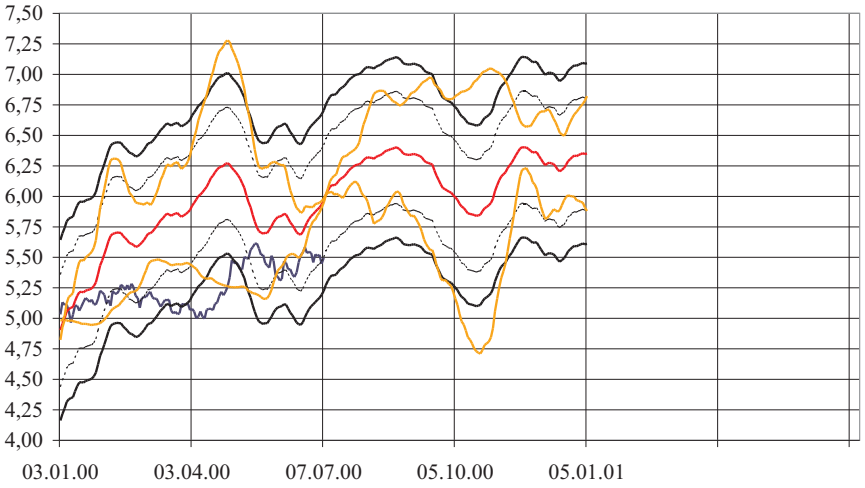


Abbildung 33: Prognose PEX-R 4 auf 6 Monate (vergrößert)

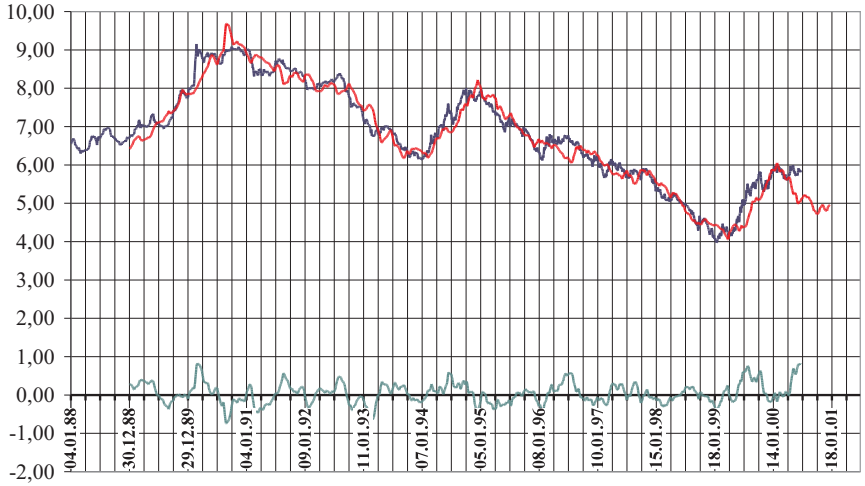


Abbildung 34: Prognose PEX-R 10 auf 6 Monate

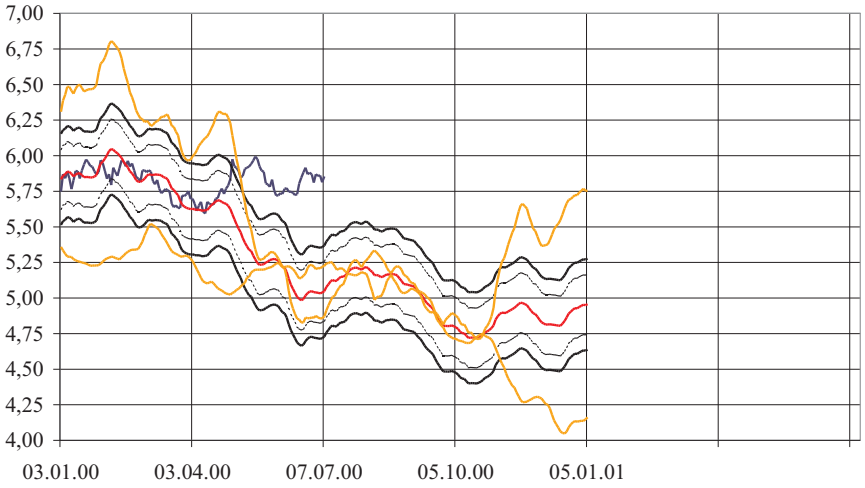


Abbildung 35: Prognose PEX-R 10 auf 6 Monate (vergrößert)

4.3.3 Prognose auf 12 Monate

n_i	12	A.T.M.	2010	Anteil	19,92%
F	1,20	A.V.M.	500	Aufteilung	450/100/225/100/...

n_h	1		2		3		4		5	
s.c.	n	j	n	j	n	j	n	j	n	j
A.G.	15	27	29	41	43	55	57	69	71	83
A.I.N.	135	519	382	1322	962	2011	1778	3329	2617	3904
A.T.N.	100	100	200	200	300	300	400	400	500	500
Verh.	1,35	5,19	1,91	6,61	3,21	6,70	4,45	8,32	5,23	7,81
$\emptyset\, t_{e.t.N.}$	0,26	1,03	0,50	1,49	0,70	2,07	1,32	2,90	1,92	3,31
ε_t^*	75,17	10,19	16,97	8,53	15,04	7,28	9,42	7,51	8,63	7,03
ε_v^*	37,49	11,58	18,56	8,48	17,60	8,24	7,70	7,75	8,57	6,34
$\emptyset\, \varepsilon_t^*$	0,273	0,101	0,130	0,092	0,122	0,085	0,097	0,086	0,093	0,084
$\emptyset\, \varepsilon_v^*$	0,193	0,107	0,136	0,092	0,132	0,091	0,088	0,088	0,092	0,079
$\emptyset\, \varepsilon_t^*$unsk.	0,929	0,342	0,441	0,313	0,415	0,289	0,329	0,294	0,315	0,284
$\emptyset\, \varepsilon_v^*$unsk.	0,656	0,364	0,461	0,312	0,449	0,307	0,297	0,298	0,314	0,270

Tabelle 28: Trainingsergebnisse der Prognose von PEX-R 1 auf 12 Monate

n_i	12	A.T.M.	2010	Anteil	19,92%
F	1,20	A.V.M.	500	Aufteilung	450/100/225/100/...

n_h	1		2		3		4		5	
s.c.	n	j	n	j	n	j	n	j	n	j
A.G.	15	27	29	41	43	55	57	69	71	83
A.I.N.	126	282	341	1311	814	2770	1572	4746	2916	6984
A.T.N.	100	100	200	200	300	300	400	400	500	500
Verh.	1,26	2,82	1,71	6,56	2,71	9,23	3,93	11,87	5,83	13,97
$\emptyset\, t_{e.t.N.}$	0,28	0,45	0,31	1,14	0,48	1,94	0,85	2,76	1,67	4,05
ε_t^*	139,27	53,58	46,23	35,95	40,30	34,41	33,91	33,59	34,93	35,88
ε_v^*	151,41	62,62	46,23	40,62	36,14	39,40	33,01	40,18	22,94	41,82
$\emptyset\, \varepsilon_t^*$	0,372	0,231	0,214	0,189	0,200	0,185	0,184	0,183	0,186	0,189
$\emptyset\, \varepsilon_v^*$	0,388	0,250	0,214	0,201	0,190	0,198	0,181	0,200	0,151	0,204
$\emptyset\, \varepsilon_t^*$unsk.	1,029	0,638	0,593	0,523	0,554	0,512	0,508	0,505	0,515	0,522
$\emptyset\, \varepsilon_v^*$unsk.	1,073	0,690	0,593	0,556	0,524	0,547	0,501	0,553	0,418	0,564

Tabelle 29: Trainingsergebnisse der Prognose von PEX-R 4 auf 12 Monate

n_i	12	A.T.M.	2010	Anteil	19,92%
F	1,20	A.V.M.	500	Aufteilung	450/100/225/100/...

n_h	1		2		3		4		5	
s.c.	n	j	n	j	n	j	n	j	n	j
A.G.	15	27	29	41	43	55	57	69	71	83
A.I.N.	130	314	320	619	680	880	1142	1109	2897	1149
A.T.N.	100	100	200	200	300	300	400	400	500	500
Verh.	1,30	3,14	1,60	3,10	2,27	2,93	2,86	2,77	5,79	2,30
$\emptyset\, t_{e.t.N.}$	0,25	1,53	0,31	2,24	0,42	3,10	0,73	5,09	1,06	7,51
ε_t^*	122,69	67,80	59,42	45,46	51,42	56,02	49,91	42,11	38,81	74,54
ε_v^*	76,00	75,70	65,41	48,46	57,17	57,32	45,44	37,66	45,09	82,85
$\emptyset\, \varepsilon_t^*$	0,349	0,260	0,243	0,213	0,226	0,236	0,223	0,205	0,197	0,272
$\emptyset\, \varepsilon_v^*$	0,275	0,274	0,255	0,220	0,239	0,239	0,213	0,194	0,212	0,287
$\emptyset\, \varepsilon_t^*$unsk.	0,688	0,511	0,479	0,419	0,445	0,465	0,439	0,403	0,387	0,536
$\emptyset\, \varepsilon_v^*$unsk.	0,541	0,540	0,502	0,432	0,470	0,470	0,419	0,381	0,417	0,565

Tabelle 30: Trainingsergebnisse der Prognose von PEX-R 10 auf 12 Monate

	Stützbereich			Generalisierung		
PEX-R	1	4	10	1	4	10
R.T.Q.	88,20%	79,47%	95,17%	88,25%	92,69%	93,73%
60%	0,31	0,45	0,35	0,59	0,35	0,49
80%	0,49	0,75	0,48	0,72	0,51	0,67
M.A.F.	0,275	0,411	0,307	0,479	0,274	0,401
M.Q.F.	0,119	0,272	0,146	0,366	0,115	0,218
r	0,990	0,953	0,949	0,847	0,911	0,845
R^2	0,980	0,908	0,901	0,718	0,830	0,714
U_d	0,207	0,286	0,282	0,335	0,156	0,250
U_p	0,255	0,403	0,386	0,538	0,248	0,400

Tabelle 31: Prognosegüten PEX-R 1, 4 und 10 auf 12 Monate

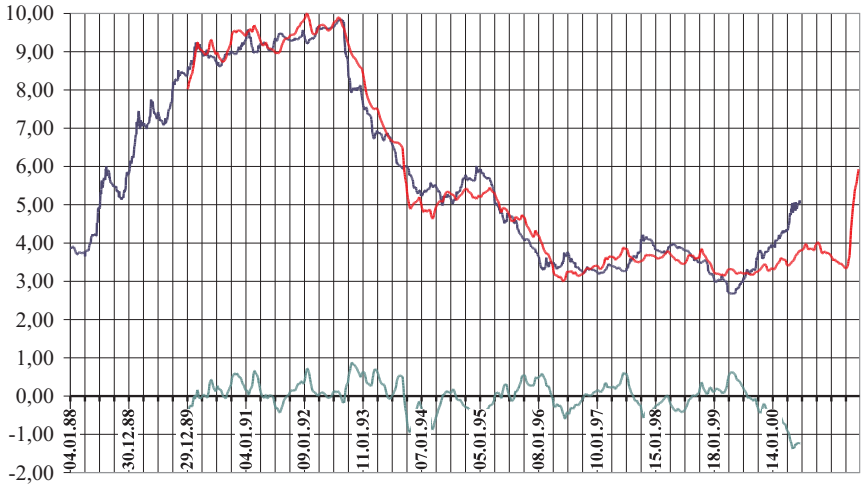


Abbildung: 36: Prognose PEX-R 1 auf 12 Monate

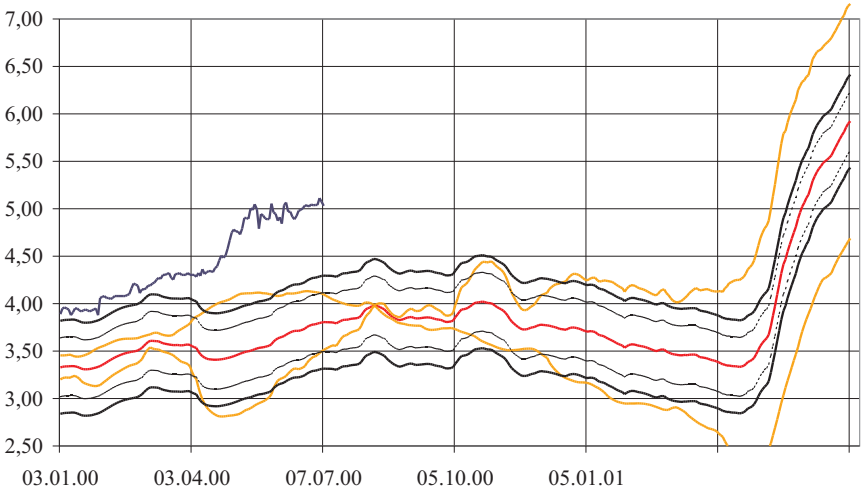


Abbildung: 37: Prognose PEX-R 1 auf 12 Monate (vergrößert)

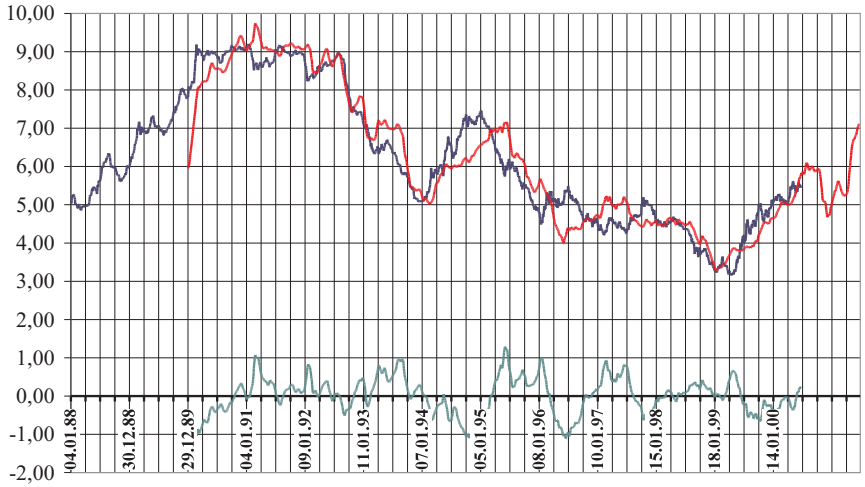


Abbildung: 38: Prognose PEX-R 4 auf 12 Monate

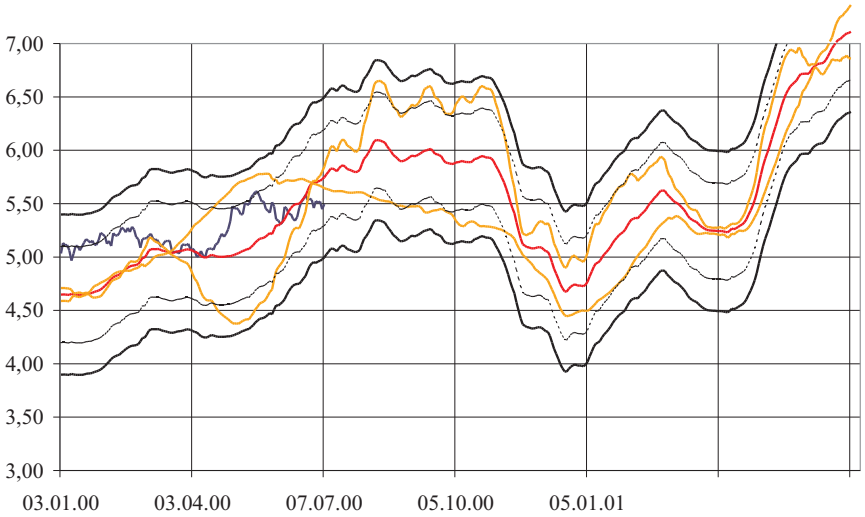


Abbildung: 39: Prognose PEX-R 4 auf 12 Monate (vergrößert)

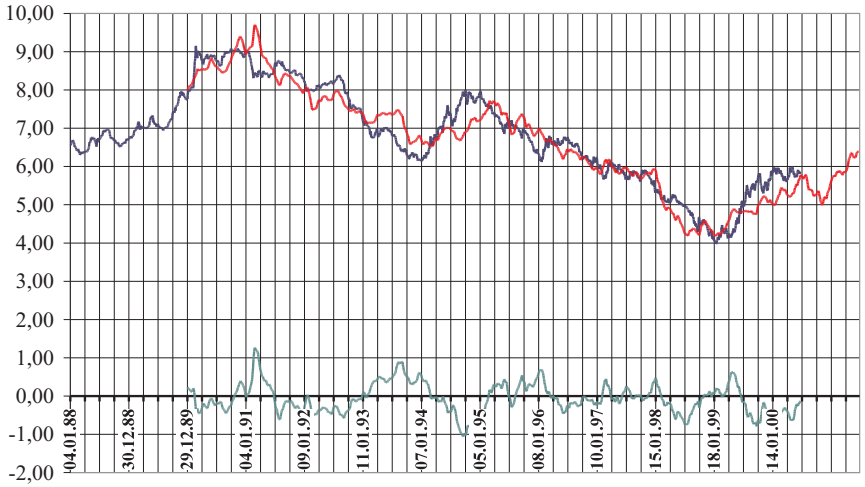


Abbildung: 40: Prognose PEX-R 10 auf 12 Monate

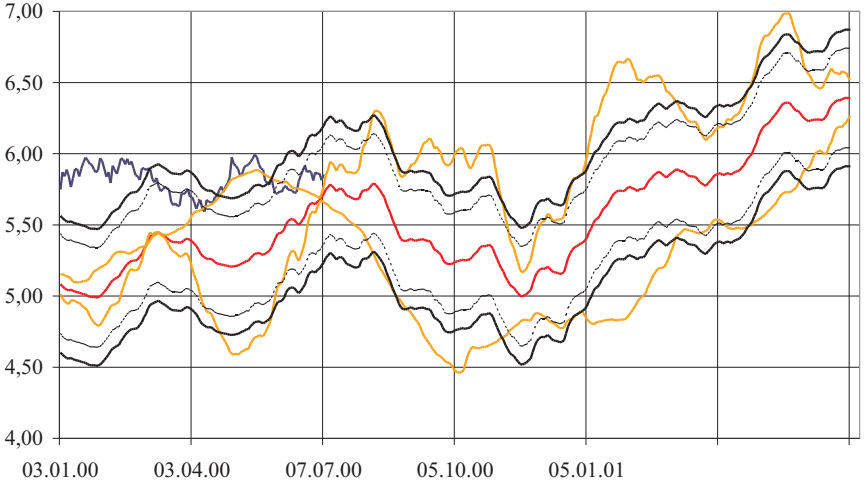


Abbildung: 41: Prognose PEX-R 10 auf 12 Monate

5 Zusammenfassung und Ausblick

Neuronale Netze können als universelle Approximatoren in zahlreichen Gebieten der Klassifikation, Identifikation, Interpolation und Prognose eingesetzt werden, wobei sie sich bei vielen Aufgabenstellungen etabliert und bewährt haben. Durch die stark vereinfachte Nachahmung neurophysiologischer paralleler Informationsverarbeitung können Neuronale Netze zur Lösung von Problemen eingesetzt werden, die mit klassischen Rechnermethoden nicht oder nur schwer zu bewältigen sind. Hervorzuheben sind vor allem die Toleranz gegenüber unvollständigem und verrauschtem Datenmaterial und die Fähigkeit, auch nichtlineare Zusammenhänge erkennen zu können. Das hier vorgestellte dreilagige Mehrschichtperzeptron kann mathematisch durch eine Funktion $f : \mathbf{R}^{n_i} \rightarrow \mathbf{R}^{n_o}$ dargestellt werden, welche die n_i Eingabewerte auf n_o Ausgabewerte abbildet. Diese Netzfunktion wird bestimmt durch die Gewichte, die den Verbindungen zwischen den Neuronen zugeordnet werden. Das Training eines Neuronalen Netzes besteht in der Suche einer geeigneten Zusammenstellung von Gewichten, so daß beim Vorliegen bekannter Ein- und Ausgabewerte der Fehler zwischen den Netzausgaben und den tatsächlichen Werten möglichst klein ist. In diesem Zusammenhang sei die Problematik des Übertrainings angesprochen, das bei einer zu starken Minimierung des Fehlers auftritt und der gewünschten Generalisierungsfähigkeit Neuronaler Netze entgegensteht. Die Vermeidung des Übertraining ist eine der wichtigsten Fragen bei der Anwendung Neuronaler Netze und wird hier durch Validierung gelöst.

Innerhalb des betrachteten Gewichtsraumes gibt es oft tausende geeigneter lokaler Minima des Trainingsfehlers, so daß sich die Suche schwierig gestaltet. Das Programm FAUN löst dieses Problem durch das Training von vielen Netzen, deren Gewichte zufällig initialisiert werden. Ausgehend von diesen verstreuten Startpunkten im Gewichtsraum wird der Trainingsfehler mittels angepaßter SQP-Verfahren minimiert. Abhängig von der Anzahl und Beschaffenheit der Trainingsmuster sowie der Netztopologie kann die Rechenzeit leicht im Stunden- bis Tagesbereich liegen. Daher besteht ein besonderes Interesse an Techniken zur Beschleunigung des Netztrainings, wobei zur Zeit im wesentlichen zwei Ansätze verfolgt werden. Zum einen ist dies der Einsatz des SYNAPSE 3 Neurocomputer-PCI-boards, mit dem Matrixoperationen um ein Vielfaches schneller als bei üblichen Rechnern durchgeführt werden können. Der zweite, hier für FAUN erstmals vorgestellte Weg, ist das parallele Training auf inhomogenen und dezentralen Rechnerkonfigurationen unter Verwendung des Softwarepakets PVM. Die grobkörnige Parallelisierung durch das entstandene FAUN-PVM Programmpaket auf Basis einer Master/Slave-Struktur ermöglicht Beschleunigungen des Trainings, die nahe an dem Faktor der zusätzlich eingesetzten Rechenleistung liegen. Bezüglich der möglichen Lastverteilungen hat sich eindeutig die dy-

namische Variante als beste Möglichkeit herausgestellt, die durch variable Gestaltung der Trainingsaufträge noch optimiert werden kann.

Bei der Entwicklung von FAUN-PVM standen neben dem Ziel einer möglichst guten Beschleunigung auch die weitgehende Konfigurierbarkeit auf persönliche Gegebenheiten und Bedürfnisse bei gleichzeitiger Fehlertoleranz im Vordergrund. Verbesserungsmöglichkeiten bieten sich hier noch in der Bedienung und Einrichtung von FAUN-PVM. So müssen einige Einstellungen noch eigenhändig auf ihre Konsistenz überprüft werden, und werden bei widersprüchlichen Angaben nicht automatisch abgefangen. Da der Quellcode von FAUN in FORTRAN 77 geschrieben ist, sind keine dynamischen Dimensionierungen der verwendeten Matrizen möglich, so daß bei Verwendung unterschiedlicher Trainingsmuster oder anderer Topologien eine zeitaufwendige Neukompilation der Slaves nötig ist. Ein wesentlicher Bestandteil von FAUN-PVM stellt daher das Shellskript SETUP_S dar, welches die Automatisierung des komplexen Einrichtungsvorgangs ermöglicht. Bei der Weiterentwicklung von FAUN-PVM sollte dennoch die Portierung der FAUN-Unterroutine nach C in Erwägung gezogen werden, weil durch die dann möglichen dynamische Dimensionierungen der Beschleunigungseffekt noch besser genutzt werden kann, der Bedienungskomfort wesentlich gesteigert und zahlreiche Fehlerquellen eliminiert werden können. FORTRAN 90 ermöglicht zwar auch dynamische Dimensionierungen, ist aber weniger empfehlenswert, da hier frei verfügbare Compiler weniger verbreitet sind.

Die parallelisierte Version von FAUN wurde schließlich bei der Entwicklung eines Zinsprognosemodells zur Vorhersage der Renditen des deutschen Pfandbriefindex PEX verwendet. Das Interesse an zuverlässigen Prognosen finanzwirtschaftlicher Zeitreihen ist offensichtlich, da auf ihrer Basis Entscheidungen getroffen werden können, die Gewinnsteigerungen und Verlustminimierungen ermöglichen. Bezüglich der Prognoseart zeigte sich die eindeutige Überlegenheit der Delta- gegenüber der Punktprognose. Sinnvollerweise sollten Expert-council-Topologien mit Netzen von einem bis fünf Neuronen verwendet werden, wobei der Einsatz von Shortcuts problemabhängig entschieden werden muß. Die rechenintensiveren komplexeren Topologien zeigten trotz der Anforderung von mehr erfolgreich zu trainierenden Netzen keine entscheidenden Verbesserungen des Trainingsfehlers. Von der Verwendung einzelner Topologien zur Prognose wird abgeraten, da diese unregelmäßige Phasen guter und schlechter Anpassung zeigten, während andererseits durch die Zusammenschaltung mehrerer Topologien unerwünschte Oszillationen gedämpft und die Generalisierungsfähigkeit steiferer Netzfunktionen genutzt werden kann.

Ein grundlegender Bewertungsmaßstab bei der Beurteilung von Prognosegüten ist das Schlagen der naiven Prognose. Dies ist eine durchaus nichttriviale Forderung, wie sich in dieser Arbeit bestätigte. Trotz großer Anstrengungen konnte der naive Anteil der Prognose nicht vollständig entfernt, sondern nur auf ein tolerierbares Maß reduziert werden.

Das vorrangige Ziel bei der Suche nach geeigneten Prognosemodellen stellt daher meines Erachtens die Dämpfung oder gar Eliminierung des naiven Anteils dar. Eine Weiterentwicklung des bestehenden Prognosemodells sollte sich auf die Wahl und Darstellung der Eingabegrößen konzentrieren, die entscheidenden Einfluß auf das Prognoseverhalten haben. In dieser Arbeit wurde hauptsächlich mit Technischen Indikatoren gearbeitet, aber auch die einfacheren Differenzmodelle zeigen ein gutes Prognosepotential. Ein Hilfsmittel bei der Beurteilung und Auswahl von Eingabegrößen stellt die Gewichtsanalyse dar, die allerdings nicht als absoluter Maßstab verwendet werden kann. Stattdessen sollten Möglichkeiten untersucht werden, unter Verwendung partieller Ableitungen der Eingabegrößen nach der Ausgabe die entscheidenden Einflußfaktoren zu finden.

Als Quintessenz und Abschluß dieser Diplomarbeit mag eine Erkenntnis von Wilhelm Busch stehen, die ich nach der intensiven Arbeit an und mit Zinsprognosemodellen nur bestätigen kann und die in weiteren Untersuchungen zur Prognose mit Neuronalen Netzen erst noch vollständig widerlegt werden muß:

> "Erstens kommt es anders, und zweitens als man denkt."

Optimierung von Warteschlangensystemen in Call Centern auf Basis von Kennzahlen-approximation

Frank Köller, Michael H. Breitner

Institut für Wirtschaftsinformatik, Universität Hannover, Königsworther Platz 1, 30167 Hannover, {koeller;breitner}@iwi.uni-hannover.de

Abstract

In diesem Aufsatz wird der Fragestellung nachgegangen, ob neuronale Netze in der Lage sind Kennzahlen für Warteschlangensysteme zu approximieren. Da für die meisten in der Praxis vorkommenden Warteschlangenprobleme keine exakten, expliziten Lösungen für die Warteschlangenkennzahlen existieren, werden diese entweder mit aufwendigen, diskreten Simulationen gelöst, oder aber das Grundproblem wird soweit vereinfacht, dass es analytisch lösbar wird. Im Gegensatz dazu muss für das Training neuronaler Netze nicht die Struktur des Problems verändert werden. Weiterhin brauchen auch nur wenige Simulationspunkte gegenüber einer „flächendeckenden“ Auswertung mit einer Simulation generiert werden, da das unvermeidliche Rauschen in den Simulationsdaten durch die kontinuierliche, approximierte Lösung geglättet wird, d. h. die Kennzahlen genauer verfügbar sind. Aufgrund deutlich weniger Simulationen besteht ein erheblicher Zeitvorteil, denn der zusätzliche Schritt des Trainings der neuronalen Netze dauert i. d. R. nur wenige Sekunden.

Anhand von Simulationen für Inbound-Call-Center wird gezeigt, dass künstliche neuronale Netze Kennzahlen von Warteschlangenproblemen, bei denen analytische Lösungen existieren, sehr gut approximieren können. Dieser Aufsatz bildet also die Grundlage dafür, dass in einem weiteren Schritt künstliche neuronale Netze auch auf allgemeine Warteschlangenprobleme angewendet werden können, für die keine exakten, expliziten Lösungen für die Warteschlangenkennzahlen existieren[1].

[1] Meist können obere und untere Schranken bestimmt werden, die die Bandbreiten für Warteschlangenkennzahlen begrenzen. Somit ist überprüfbar, ob die approximierten Kennzahlen innerhalb dieser Bandbreiten liegen.

1 Einleitung

Kundenservice Center bilden die Schnittstelle zwischen Unternehmen und Kunden und haben somit eine Schlüsselposition inne: Von hier aus werden Geschäftsbeziehungen aufgebaut, gesteuert und ausgebaut, sowohl im B2B- als auch im B2C-Bereich. Insbesondere in den Unternehmen, wo heute bereits 90 % aller Kundenkontakte im Kundenservice Center abgewickelt werden, kommt dem Call Center eine nicht zu unterschätzende Bedeutung für den Gesamterfolg des Unternehmens zu. Es ist daher unabdingbar, die Abläufe im Call Center permanent im Blick zu haben und zu verbessern. Maßgebliche Erfolgsfaktoren sind hierbei Kosten und Performance. Der überwiegende Anteil, etwa dreiviertel des Gesamtbudgets, in einem Call Center sind personalbezogene Ausgaben (Call Center-Benchmark Kooperation 2004). In der Praxis erfolgt gegenwärtig die Personalbedarfsermittlung und -einsatzplanung in der Regel in den folgenden drei Schritten (vgl. Helber und Stolletz 2004):

1. Prognose des Anrufaufkommens je Periode (häufig 30- oder 60-Minutenintervalle).
2. Ermittlung der erforderlichen Zahl von Agenten je Periode für einen vorgegebenen Servicegrad hinsichtlich der Wartezeit (meist mit dem M/M/*c*-Modell).
3. Zeitliche Einplanung der Mitarbeiter über die Perioden (oder zeitliche Einplanung „anonymer“ Schichten mit anschließender Zuordnung der Mitarbeiter zu den Schichten).

An die Personaleinsatzplanung im Schritt 3 schließt sich noch eine Echtzeit-Steuerung an, in der in Abhängigkeit des aktuellen Systemzustandes z. B. die Pausen der Agenten, Besprechungen oder Trainingsmaßnahmen zeitlich festgelegt werden.

Im ersten Schritt, der Prognose, ist ein Anrufaufkommen vorherzusagen, das zwar innerhalb eines Tages oder einer Woche hochgradig variabel ist, dabei aber häufig wiederkehrende Muster aufweist (vgl. Abbildung 1). Die Datengrundlage für die Prognose wird dabei in der Regel von der automatischen Anrufverteilungsanlage (Automatic call distribution (ACD)-Anlage) geliefert. Relativ einfache Prognoseverfahren sind die exponentielle Glättung erster Ordnung auf Basis korrespondierender Zeitabschnitte oder eine Prognose durch gleitende Mittelwerte. Zieht man aufwendigere ARIMA-Methoden heran, vgl. Box et al. (1994), so erhält man bessere Ergebnisse. Wir werden uns in dieser Arbeit auf den Schritt 2 beschränken und Rückschlüsse auf eine mögliche Einsatzplanung an dem konkreten

Beispiel eines Inbound-Call-Centers machen[2]. Das folgende Kapitel gibt einen allgemeinen Überblick zu Call Centern. In der Praxis wird in Call Centern meist noch das M/M/*c*-(oder „Erlang-C“-)Warteschlangenmodell, welches in Kapitel 3 erläutert wird, bei der Personaleinsatzplanung eingesetzt. Da für das M/M/*c*-Modell analytische Lösungen für alle Kennzahlen existieren wird die mathematische Analyse von mit dem Neurosimulator FAUN[3] approximierten Kennzahlen möglich[4]. Dies geschieht nach einer kurzen Einführung in die künstliche Intelligenz in Kapitel 4 und der Erläuterung in Kapitel 5, wie die Simulationsdaten für das Training der neuronalen Netze generiert werden, in Kapitel 6.

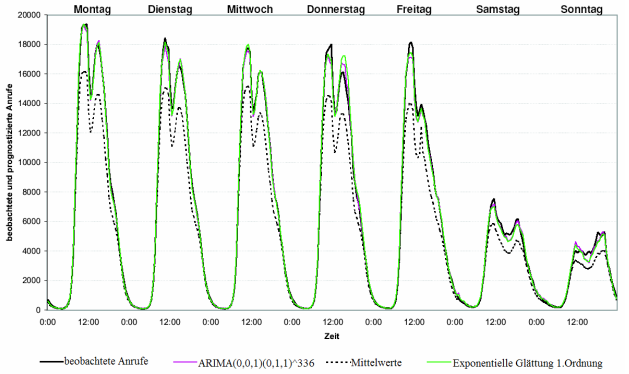


Abb. 1. Anrufaufkommen und Prognosen in Halbstundenintervallen in den Call Centern des Auskunftsdienstes der Deutschen Telegate AG vom 2. – 8.11.1998. Deutlich sind die Auswirkung der Mittagspausen und des Wochenendes zu erkennen (Helber und Stolletz 2004).

[2] In Helber und Stolletz (2004) wird eine gewinnmaximierende Agentenallokation vorgestellt, bei der gewissermaßen als „Nebenprodukt“ entsprechende Wartezeitmaße ermittelt werden. Hier wird dagegen nur das M/M/*c*-Modell betrachtet.

[3] „**F**ast **A**pproximation with **U**niversal **N**eural Networks“. Neurosimulator bezieht sich nicht auf die Simulation von Warteschlangen, sondern auf die komfortable, GUI-unterstützte Simulation gehirnanaloger Vorgänge, die als Training bzw. Lernen von künstlichen neuronalen Netzen bekannt sind (Breitner 2003).

[4] Für das M/M/1-Modell teilweise untersucht in Barthel (2003), einer Diplomarbeit betreut durch die Autoren.

2 Beispiel Call Center

In einem Call Center werden organisatorisch Telefonarbeitsplätze in Verbindung mit informations- und kommunikationstechnischer Unterstützung in Großraumbüros zusammengefasst. Die Mitarbeiter, welche in koordinierte Gruppen eingeteilt und auf die Durchführung von Telefongesprächen spezialisiert sind, werden auch als Agenten bezeichnet. Ziel eines jeden Call Centers ist ein verbesserter Kundenkontakt bzw. die Kundenbetreuung und -gewinnung bei gleichzeitiger Optimierung der Wirtschaftlichkeit. Dementsprechend wird ein Call Center als Dienstleistungsbetrieb bezeichnet, bei dem der Produzent des Dienstes und der Konsument zwar räumlich voneinander getrennt, aber zeitlich in der Regel aneinander gebunden sind. Stehen dem Agenten neben dem Telefon noch mehrere Kommunikationskanäle zur Verfügung, nennt man dies auch Contact Center oder Kundenservice Center.

Grundsätzlich werden hereinkommende Anrufe als Inbound-Gespräche und ausgehende Anrufe als Outbound-Gespräche bezeichnet. Entsprechend können Call Center in Inbound- und Outbound-Call-Center bzw. Mischformen unterteilt werden[5].

2.1 Call-Center-Marktentwicklung

Über die Call-Center-Marktentwicklung gibt es unterschiedliche Meinungen. Beispielsweise prognostizieren die Analysten von Datamonitor (Datamonitor 2004), dass der Call-Center-Markt weiter rasant wächst: Die Zahl der Call Center soll in Deutschland fortlaufend steigen, doch ebenso kontinuierlich die Zahl der Beschäftigten je Call Center sinken. Trotzdem sollen hier unter dem Strich viele Arbeitsplätze entstehen[6] (vgl. Abbildung 2). Allerdings warnen die Analysten auch davor, dass immer mehr Call Center nach Polen, Tschechien oder Ungarn abwandern, wo qualifiziertes und kundenfreundliches Personal zu niedrigeren Kosten bereit stehe. In der Call Center Benchmarkstudie 2003 wurde hingegen gezeigt, „…dass die Anforderungen des Marktes beinahe alle Betreiber vor die gleichen Probleme und Schwierigkeiten stellen. Inhouse-Center und Dienstleister haben gleichermaßen mit den Auswirkungen zu kämpfen, die wirtschaftlicher Stillstand, Kostendruck und dennoch hohe Service-Erwartungen mit sich

[5] In diesem Artikel beziehen wir uns nur auf die Inbound-Call-Center.

[6] Es ist darauf zu achten, dass zwischen der Zahl der Beschäftigten und der Zahl der Arbeitsplätze genau differenziert wird, da Call Center i. d. R. einen hohen Anteil an Teilzeitkräften einsetzen.

bringen. Die einstige „Boom"-Branche, in der „maximaler" Service ohne Rücksicht auf die Kosten geboten wurde, expandiert nicht mehr, sondern konzentriert sich mit den vorhandenen Kapazitäten auf den Versuch, sich den veränderten Rahmenbedingungen anzupassen" (Kestling 2004).

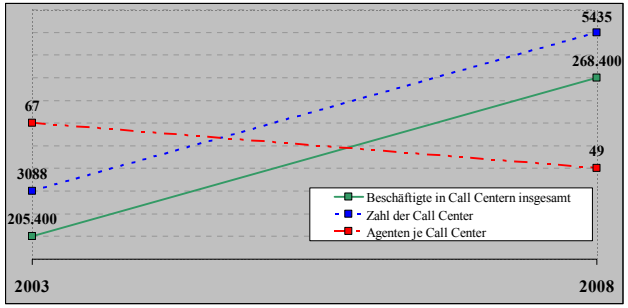


Abb. 2. Die Zahl der Call Center steigt in Deutschland fortlaufend, doch ebenso kontinuierlich sinkt die Zahl der Beschäftigten je Call Center (Datamonitor 2004).

2.2 Steigender Kostendruck in Call Centern

Das steigende Kommunikationsaufkommen in den beiden vergangenen Jahren und unreflektierter maximaler Service in Call Centern verursachten eine Kostenexplosion, die in keinem proportionalen Verhältnis zur Umsatzentwicklung steht (Call Center-Benchmark Kooperation 2004). Somit stehen Call Center nun vor der konkreten Aufgabe, Maßnahmen zur Kostensenkung aktiv umzusetzen. Dieses ist jedoch problematisch, da der überwiegende Anteil des Gesamtbudgets personalbezogene Ausgaben sind (Gehälter, Personalauswahl, Schulung und Training). Während dieser Kostenblock im Jahre 1998 noch mit rund 61% des Gesamtbudgets beziffert wird (Henn et al. 1998, S. 99), ist der Wert laut der Benchmarkstudie im Jahre 2003 schon auf rund 75% gestiegen. Die verbleibenden Positionen (Miete, lfd. Betriebskosten, Ausstattung, etc.) weisen jeweils nur eine nachgeordnete Größenordnung auf und können in der Praxis auch nicht weiter gesenkt werden. Somit sind nunmehr Ansätze gefordert, das angebotene Servicespektrum an die tatsächlichen Bedürfnisse anzupassen und gleichzeitig die Effizienz der Prozesse bzw. die Auslastung der Agenten zu steigern. Bei der Fokussierung auf Einsparpotenziale, wie die Freisetzung der tatsächlich entbehrlichen Kapazitäten, darf das Call-Center-Manage-

ment jedoch nicht die notwendige Kundenzufriedenheit gefährden (vgl. Abbildung 3).

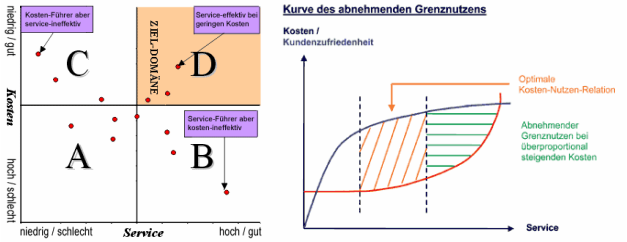


Abb. 3. Ziel eines jeden Call Centers ist es einen guten Service bei möglichst geringen Kosten anzubieten (links). Je höher aber der angebotene Service (und damit auch die Kundenzufriedenheit) ist, desto höher sind die hierfür aufzuwendenden Kosten (rechts) (Call Center-Benchmark Kooperation 2004).

3 Warteschlangentheorie anhand eines Inbound-Call-Centers

Die Warteschlangentheorie beschäftigt sich mit den strukturellen Zusammenhängen innerhalb von Warteschlangensystemen. Sie sucht nach mathematischen Lösungen um die Kennzahlen von Warteschlangensystemen berechnen zu können. Das erste Mal wurde die Warteschlangentheorie im Jahre 1908 durch die optimale Dimensionierung von Telefonnetzen durch A. K. Erlang bekannt (Zimmermann 1997, S. 362). Mit der Entwicklung der Computer wurde es dann möglich, Warteschlangenprozesse zu simulieren, um für Systeme ohne analytische Lösung Kennzahlen zu ermitteln, ohne dabei einen direkten mathematischen Systemzusammenhang herzustellen.

3.1 Warteschlangensysteme

Eine Warteschlange entsteht beispielsweise, wenn Personen in einem Call Center anrufen und dort alle Agenten besetzt sind. Meist werden sie dann in einer Warteschleife abgefangen und warten solange bis der nächste Agent frei ist. Im Kontext von Warteschlangen werden alle Personen oder Jobs als Kunden und die Warteschleife als Warteschlange bezeichnet. Die

Elemente, die ein Warteschlangensystem bilden, sind in Abbildung 4 anhand eines Schalters, wie er z. B. bei einer Post vorkommt, dargestellt.

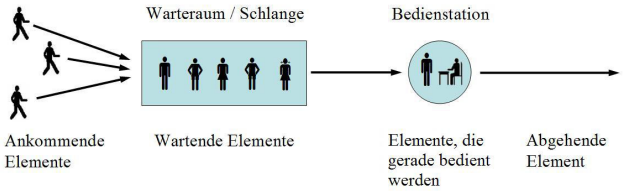


Abb. 4. Das Warteschlangensystem

Dabei sind die wichtigsten Elemente im Einzelnen:

- **Der Ankunftsprozess**. Wenn Kunden zu den Zeiten t_1 , t_2 , … , t_n eintreffen, so werden die Zeiten $\tau_j = t_j - t_{j-1}$ als Zwischenankunftszeiten bezeichnet. Es wird allgemein angenommen, dass die τ_j eine Folge von unabhängigen und identisch verteilten (iid) Zufallsvariablen sind.
- **Die Verteilung der Bedienzeit**. Die Zeit, die jede Person am Schalter bzw. im Gespräch mit dem Call Center Agenten verbringt wird ihre Bedienzeit genannt. Die Bedienzeiten werden ebenfalls als unabhängige, identisch verteilte Zufallsvariablen angenommen.
- **Anzahl an Bedieneinheiten**. Oft arbeiten in einem Call Center mehrere Agenten, die alle dieselben Dienste anbieten. In diesem Fall spricht man von mehreren Bedieneinheiten. Bieten die Agenten jedoch verschiedene Dienste an, so werden sie in Gruppen mit gleichem Angebot gegliedert, die dann jeweils eine Warteschlange bilden[7].

Die Kapazität der Warteschleife ist in Call Centern begrenzt, d. h. wenn die Warteschleife voll ist, werden weitere Anrufer abgewiesen bzw. erhalten ein Besetztzeichen. Dennoch wird zur Vereinfachung der Berechnung der Personaleinsatzplanung in Call Centern eine unbegrenzte Warteschlange angenommen. Ebenso ist bei dieser Berechnung die Anzahl aller potentiellen Kunden (Population) unendlich und die Kunden werden in der Reihenfolge bedient, in der sie ankommen (First Come, First Served (FCFS)).

[7] Ausführliche Darstellungen zur Warteschlangentheorie findet man z.B. in Schassberger (1973), Bolch (1989), Meyer und Hansen (1996, S. 210 ff.) oder Hillier und Lieberman (1997, S. 502 ff.)

Sind zusätzlich noch der Ankunftsprozess poissonverteilt, d. h. die Zwischenankunftszeiten sind iid und exponentialverteilt und die Bedienzeit exponentialverteilt, so wird dies als M/M/*c*-Modell bezeichnet. Dabei stehen die beiden „M“ für „Markovian“ und entsprechen den Exponentialverteilungen der Zwischenankunftszeiten und der Bedienzeit. *c* ist hierbei die Anzahl der Bedieneinheiten.

3.2 Das M/M/*c*-System und ein Inbound-Call-Center

Die in der Praxis eingesetzte Personaleinsatzplanungssoftware zieht regelmäßig das so genannte M/M/*c*- (oder “Erlang-C”-) Warteschlangenmodell heran, mit dem a priori unter bestimmten Annahmen zum einen

- die Wahrscheinlichkeit $P(W \le t)$, dass die zufällige Wartezeit W nicht länger als t Zeiteinheiten ist, und zum anderen
- die mittlere Wartezeit $E(W)$ der Anrufer

berechnet werden kann. In diesem Modell wird unterstellt, dass in dem Call Center Anrufe mit der durchschnittlichen Rate λ eingehen und jeder der c identischen Agenten Anrufe mit einer durchschnittlichen Rate μ bearbeitet. Die Zwischenankunftszeiten seien ebenso wie die Bearbeitungszeiten unabhängig exponentialverteilt, der Warteraum unendlich groß und alle Anrufer geduldig. Unter diesen Bedingungen ist das System stabil in dem Sinn, dass die Anzahl der Anrufer im System nicht über alle Grenzen steigt, wenn die Anrufrate λ strikt kleiner ist als die kombinierte Bearbeitungsrate (oder -geschwindigkeit) $c\mu$ aller c Agenten:

$$\lambda < c\mu\,,\ a := \frac{\lambda}{\mu} \text{ und } \rho := \frac{1}{c}\frac{\lambda}{\mu} = \frac{a}{c}\,. \tag{1}$$

Dabei stellt a das Arbeitsvolumen in der dimensionslosen Einheit „Erlangs“ dar. Die stationären Lösungen im Allgemeinen Fall für die Kennzahlen des M/M/*c*-Systems existieren genau dann, wenn der Servicegrad $\rho < 1$ ist[8], welches hier durch die Annahme $\lambda < c\mu$ schon gegeben ist. Der stationäre Zustand von Warteschlangenprozessen ist eine wichtige Eigenschaft in der Warteschlangentheorie. Er dient, zusammen mit der Markov-Eigenschaft, als Voraussetzung dafür, dass die Kennzahlen von Warteschlangensystemen und deren Verteilungen überhaupt analytisch bestimmt werden können[9]. Eine wichtige Kenngröße für eine M/M/*c*-Warteschlange

[8] Siehe hierzu auch Kapitel 5.2.

[9] Ein stochastischer Prozess ist stationär, wenn sich der Erwartungswert und die Varianz in der Zeit nicht ändern

ist die Wahrscheinlichkeit, dass ein ankommender Kunde warten muss. Der Ausdruck dafür ist bekannt unter dem Namen Erlangsche C-Formel oder Erlangsche Warteformel. Sie ist gegeben durch

$$P(N \ge c) = \frac{\frac{a^c}{c!}}{\left(1-\frac{a}{c}\right)\sum_{n=0}^{c-1}\frac{a^n}{n!}+\frac{a^c}{c!}} =: C(c,a) \tag{2}$$

,

wobei N die Zahl der Kunden im System ist, d.h. die Zahl der Wartenden N_q plus der Zahl der Kunden, die gerade bedient werden N_s.

Die zu erwartende Zahl der Kunden in der Schlange ist

$$E(N_q) = \frac{\rho}{1-\rho}C(c,a)\,. \tag{3}$$

Die zu erwartende Wartezeit in der Schlange ist also

$$E(W_q) = \frac{1}{\lambda}E(N_q)\,. \tag{4}$$

4 Neuronale Netze

Im Idealfall lernen neuronale Netze ähnlich wie ein Gehirn an Beispielen. In künstlichen neuronalen Netzen werden einige Strukturen eines Nervensystems in karikativer Weise imitiert, um so ein Programm zu erhalten, mit dem Daten in einer bestimmten Weise verarbeitet werden können. Ein künstliches neuronales Netz besteht aus einer Menge von Knoten und deren Verbindungen untereinander, wobei jeder Knoten eine einzelne Nervenzelle modelliert. Vereinfacht ist ein neuronales Netz ein gerichteter und gewichteter Graph. Jeder Knoten j wird durch eine Variable $a_j(t)$ zum Zeitpunkt t beschrieben, die seinen Aktivierungszustand anzeigt. Für jede Verbindung zwischen zwei Knoten wird eine weitere Variable w_{ij} eingeführt, die die Stärke der Verbindung zwischen den Nervenzellen modelliert und als das Gewicht von Neuron i nach Neuron j bezeichnet wird (vgl. Abbildung 5).

4.1 Neurosimulator FAUN

Die Entwicklung des Neurosimulators FAUN begann 1997 an der TU Clausthal und wird mit der FAUN-Projektgruppe an der Universität Han-

nover weitergeführt[10]. Heute ist es mit FAUN Release 1.0 komfortabel möglich, Probleme des überwachten Lernens mit künstlichen neuronalen Netzen zu lösen. Als Netze sind so genannte 3- und 4-lagige Perzeptrons und Radial-Basis-Netze mit und ohne Direktverbindungen verfügbar (vgl. Abbildung 5). Direktverbindungen zwischen der Eingabeschicht und der Ausgabeschicht erhöhen die Flexibilität eines künstlichen neuronalen Netzes. Es können „schwach nichtlineare“ Abhängigkeiten in den Ein- und Ausgabezusammenhängen leichter und besser approximiert werden. Im Vergleich zu anderen Neurosimulatoren trainiert FAUN Netze extrem schnell und konvergiert, dank globaler Optimierung, sehr zuverlässig (Breitner (2003)). Für FAUN 1.0 ist eine sehr komfortable, graphische Benutzeroberfläche unter Microsoft Windows und LINUX verfügbar. Mit der Benutzeroberfläche kann das Training der künstlichen neuronalen Netze einfach gesteuert und überwacht werden. Ferner können die besten trainierten Netze einfach ausgewählt und durch Bereitstellung des C- und FORTRAN-Quellcodes evaluiert werden.

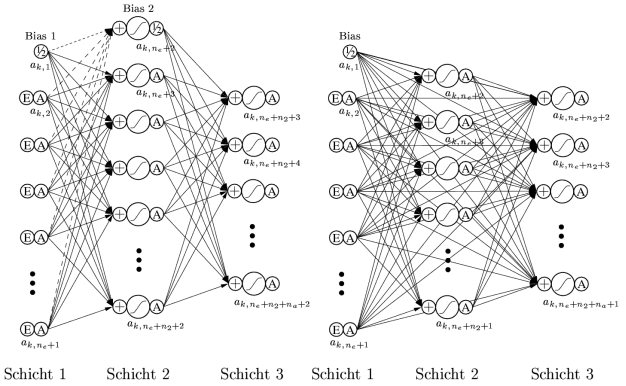


Abb. 5. Vollständig verbundenes dreilagiges Perzeptron ohne (links) bzw. mit Direktverbindungen (rechts) mit n_2 inneren Neuronen für eine n_e-dimensionale Eingabe $\boldsymbol{x}_k$ und eine n_a-dimensionale Ausgabe $f_{app}(\boldsymbol{x}_k; \boldsymbol{p})$.

[10] Siehe auch www.iwi.uni-hannover.de/faun.html.

4.2 Überwachtes Lernen

Überwachtes Lernen bedeutet, dass Ein-/Ausgabezusammenhänge $(\boldsymbol{x}_i, \boldsymbol{y}_i)$ - so genannte Muster - mit Input $\boldsymbol{x}_i \in \mathbb{R}^{n_e}$ und Soll-Output $\boldsymbol{y}_i \in \mathbb{R}^{n_a}$, $i = 1, 2, \ldots, n_m$, aus einem Musterdatensatz $\mathcal{D}_m$ gegeben sind, für die eine „möglichst gute“ $\mathcal{C}^c$-Approximationsfunktion $f_{app}(\boldsymbol{x}; \boldsymbol{p}^*)$ berechnet werden soll, wobei $f_{app}(\boldsymbol{x}; \boldsymbol{p}^*) : \mathbb{R}^{n_e} \times \mathbb{R}^{n_p} \to \mathbb{R}^{n_a}$ ist. $f_{app}(\boldsymbol{x}; \boldsymbol{p})$ hängt unendlich oft differenzierbar von $\boldsymbol{x}$ und dem wählbaren Parametervektor $\boldsymbol{p}$ ab. Dies ist u. a. wichtig für die Verwendbarkeit in der Praxis bzw. das Lösen schwieriger, multivariater Approximationsprobleme, wie z. B. Prognosen für Aktien, Indizes oder Zinsen sowie Kapitalmarktanalysen und -bewertungen (auch für Derivate). Dabei müssen die Muster in $\mathcal{D}_m$ problemgerecht auf den Trainingsdatensatz $\mathcal{D}_t := \{(\boldsymbol{x}_1, \boldsymbol{y}_1), \ldots, (\boldsymbol{x}_{n_t}, \boldsymbol{y}_{n_t})\}$ und den Validierungsdatensatz $\mathcal{D}_t := \{(\boldsymbol{x}_{n_t+1}, \boldsymbol{y}_{n_t+1}), \ldots, (\boldsymbol{x}_{n_m}, \boldsymbol{y}_{n_m})\}$ aufgeteilt werden. Wichtig ist eine Equilibrierung und Skalierung $\boldsymbol{x}_i \in [-1,1]^{n_e}$ und $\boldsymbol{y}_i \in [-c,c]^{n_a}$ mit $c \in]0,1[^+$ für alle Muster. Für das Training der neuronalen Netze wird in der Regel der Trainings- und Validierungsfehler

$$\varepsilon_t(\boldsymbol{p}) := \sum_{i=1}^{n_t} \sum_{k=1}^{n_a} \left(f_{app_k}(\boldsymbol{x}_i; \boldsymbol{p}) - y_{i,k} \right)^{2q},$$
$$\varepsilon_v(\boldsymbol{p}) := \sum_{i=n_t+1}^{n_m} \sum_{k=1}^{n_a} \left(f_{app_k}(\boldsymbol{x}_i; \boldsymbol{p}) - y_{i,k} \right)^{2q} \tag{5}$$

benutzt, wobei $q \in \mathbb{N}$ gelten muss und oft $q = 1$ verwendet wird.

Eine gute Approximationsfunktion $f_{app}(\boldsymbol{x}; \boldsymbol{p}^*)$ weist einen kleinen Fehler $\varepsilon_t(\boldsymbol{p}^*)$ pro Muster auf, d. h. $f_{app}(\boldsymbol{x}; \boldsymbol{p}^*)$ synthetisiert die Ein-/Ausgabezusammenhänge aus $\mathcal{D}_t$ ausreichend genau. Darüber hinaus ist ein gutes globales Approximations- bzw. Extrapolationsverhalten von $f_{app}(\boldsymbol{x}; \boldsymbol{p}^*)$ erforderlich. Dafür ist notwendig, dass auch der Fehler $\varepsilon_v(\boldsymbol{p}^*)$ pro Muster klein ist. In der Praxis muss $f_{app}(\boldsymbol{x}; \boldsymbol{p}^*)$ noch weiteren Anforderungen genügen, wie z. B. eine kleine Maximal- oder Gesamtkrümmung aufweisen (Breitner 2003).

5 Simulation von Warteschlangenmodellen

Simulation ist „der experimentelle Zweig des Operations Research“ (Hillier und Lieberman 1997). Komplexe Zusammenhänge werden auf dem

Rechner nachgespielt, weil Ausprobieren in der Realität oft zu teuer ist oder das Objekt dabei zerstört wird. Beispielsweise werden im Flugsimulator kritische Turbulenzen untersucht.

Simulation kann auch dann verwendet werden, wenn es keine (exakten) mathematischen Lösungsverfahren gibt, oder wenn es zwar prinzipiell mathematische Lösungsmöglichkeiten gibt, diese jedoch zu kompliziert sind. Oft erfordern mathematisch exakte Lösungen zudem einschränkende Annahmen. Etwa bei der Untersuchung von stochastischen Zufallseinflüssen, wie z. B. dem Wartesystem M/M/*c*.

5.1 Simulation des Inbound-Call-Centers

Für ein Inbound-Call-Center sind einschränkende Annahmen bei dem M/M/*c*-Wartesystem, dass die Zwischenankunftszeiten ebenso wie die Bearbeitungszeiten unabhängig exponentialverteilt seien, alle Anrufer geduldig sind und der Warteraum unendlich groß sei. Auf viele Inbound-Call-Center treffen diese Annahmen des Erlang-C-Modells eher nicht zu. Meist steht nur eine begrenzte Zahl an Wartepositionen zur Verfügung, das heißt, wenn dieser Warteraum voll ist, erhält der Anrufer ein Besetztzeichen. Meist weisen Call Center mehrere Klassen von Anrufern oder Agenten auf oder die Anrufer sind ungeduldig und legen vorzeitig auf. Sind die Zwischenankunftszeiten und die Bearbeitungszeiten nicht exponentialverteilt, so ist es nur schwer bzw. gar nicht möglich, eine analytische Lösung zu finden. Dennoch können grundlegende Zusammenhänge auf der Basis dieses einfachsten Modells in konzeptionell klarer Weise erläutert werden und so wird es regelmäßig bei der Personaleinsatzplanung in der Praxis eingesetzt.

Die etablierten verschiedenen Simulationsprachen, wie z. B. GPSS (ab 1962 entwickelt), SIMSCRIPT, SIMULA oder DYNAMO besitzen integrierte Prozeduren, die es ermöglichen einige Warteschlangenprobleme in kurzer Zeit zu modellieren. Die Prozeduren der Programme sind aber nur allgemein anwendbar und nicht direkt problemspezifisch angepasst[11]. Deshalb und weil verschiedenste komplexere Warteschlangenprobleme ohne einschränkende Annahmen simuliert werden sollen, wurde ein eigenes Simulations-Tool erst in Maple, dann in C++ entworfen. Während die Simulationen auf einem Intel Pentium 4 mit 1,8 GHz und 512 MB RAM in Maple durchaus eine Stunde betragen können, sind es bei dem C++ Programm nur wenige Sekunden[12]. Für das Maple-Tool spricht jedoch, dass jede Simulation sofort mit Maple sowohl mathematisch, als auch graphisch analysiert werden kann. Weiterhin besitzt der Neurosimulator FAUN eine Maple Schnittstelle, die es einfach ermöglicht, die neuronalen Netze nicht nur mit den Simulationspunkten zu vergleichen, sondern auch mit den exakten analytischen Lösungen der Kennzahlen. Mit den Simulationsprogrammen können alle Kennzahlen simuliert werden, wir gehen hier aber nur speziell auf die mittlere Wartezeit in der Schlange und auf die Auslastung des Systems ein (vgl. Abbildung 6 und 11), da dies die relevanten Entscheidungsvariablen für einen Call-Center-Manager zur Personaleinsatzplanung sind.

[11] Vertiefende Beispiele und Erläuterungen zu den Simulationsprogrammiersprachen sind in Zimmermann (1997, S. 338), Domschke (2002, S. 220) und Siegert (1991) zu finden.

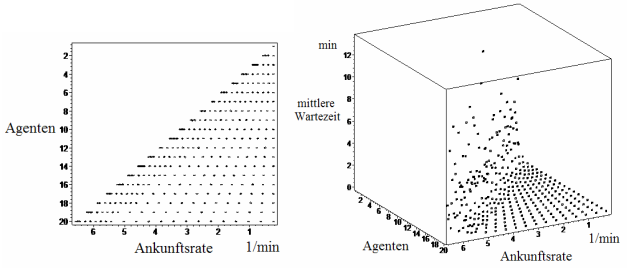


Abb. 6. Links: 349 Simulationspunkte, wobei für die Ankunftsrate $\lambda < c\mu$ gilt und $c = 1, 2, \ldots, 20$ die Anzahl der Agenten und $\mu = 1/3$ die Bedienrate ist. Rechts: Die 349 Punkte und die dazugehörige mittlere Wartezeit, wobei für jeden Punkt 5.000 ankommende Anrufer simuliert wurden.

Rechts in Abbildung 6 sind die Simulationspunkte für die mittlere Wartezeit (in Minuten) für die Kunden in der Warteschleife in Abhängigkeit von der Ankunftsrate und der Anzahl an Agenten zu sehen. Die Verteilung der Simulationspunkte in der Ebene, die aufgespannt wird durch die Anzahl der Agenten und der Ankunftsrate, ist links zu sehen. Es ist eindeutig zu erkennen, dass die Bedingung aus Gleichung (1), welche besagt, dass die Anrufrate λ strikt kleiner ist als die kombinierte Bearbeitungsrate $c\mu$ aller c Agenten, eingehalten wird. Hierbei wird angenommen, dass ein Beratungsgespräch durchschnittlich bei allen Agenten drei Minuten dauert, also

[12] Die Zeit für die Simulationen hängt direkt proportional ab von der Anzahl der simulierten Punkte und der Anzahl an Kunden, die pro Punkt simuliert werden.

die Bedienrate $\mu = 1/3$ ist und maximal 20 Agenten eingesetzt werden. Geht die Ankunftsrate gegen die Bedienrate, so ist in der Simulation zu erkennen, dass die mittleren Wartezeiten schlagartig ansteigen (vgl. Abbildung 6 rechts), während sie vorher nahezu Null sind. In dem Bereich, wo die mittlere Wartezeit nahezu Null ist, werden die Simulationspunkte durch eine variable Schrittweite bezüglich der Ankunftsrate „ausgedünnt", um nicht zu viele redundante Informationen für das Training der künstlichen neuronalen Netze zur Verfügung zu stellen und um die Zeit für die Simulationen zu senken[13]. Die Anzahl der Simulationspunkte in diesem Bereich sollte aber ungefähr genauso groß sein wie die Anzahl der übrigen Punkte, da sonst die zu approximierende Funktion hier einen zu hohen Fehler aufweist und nicht wie die analytische Lösung, bzw. auch die Simulation, eine mittlere Wartezeit von nahe Null hat (vgl. Kapitel 6).

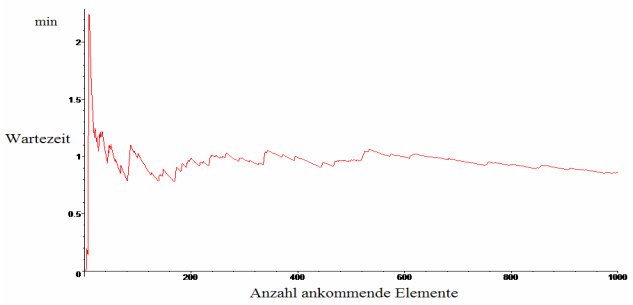


Abb. 7. Stationäres Verhalten der durchschnittlichen Wartezeit bei einer Simulation von n = 1.000 Ankünften und einer Bedienstation (ein Call Center Agent): bis n = 200 Ankünfte unterliegt das System noch starken Schwankungen, stabilisiert sich dann aber und konvergiert gegen seinen Erwartungswert.

5.2 Genauigkeit stochastischer Simulationen

Der stationäre Zustand des simulierten Systems schwankt im Zeitablauf, hat aber einen Mittelwert, um den die einzelnen Zustände schwanken bzw. zu dem sie konvergieren (vgl. Abbildung 7). Simulationen, die mit Verteilungen arbeiten, generieren Zufallsvariablen. Bei unendlich vielen Versuchen würde der Zustand des Systems durch die generierten Zufallsvariablen genau seinen Erwartungswert $E(x)$ treffen. Die Unendlichkeit in diesem Zusammenhang zu simulieren ist aber unmöglich. Die Genauigkeit des Erwartungswertes kann jedoch nach einer Gesetzmäßigkeit verbessert werden. Die Gesetzmäßigkeit besagt, wenn die Versuche um das n-fache steigen, verbessert sich der Fehler um das $\frac{1}{\sqrt{n}}$ -fache. Wenn der Fehler also auf nur noch 1/10 verbessert werden soll, müssen die Versuche verhundertfacht werden (Siegert 1991, S. 167). Dieses Verhältnis zeigt auf, wie zeitaufwendig eine solche Simulation sein kann, ohne dass eine wesentliche Verbesserung der Genauigkeit erreicht wird.

[13] Der Zeitfaktor ist hauptsächlich von Bedeutung, wenn das Maple-Tool benutzt wird bzw. auch bei dem C++ Tool, wenn wesentlich mehr als 100.000 Ankünfte pro Simulationspunkt simuliert werden sollen.

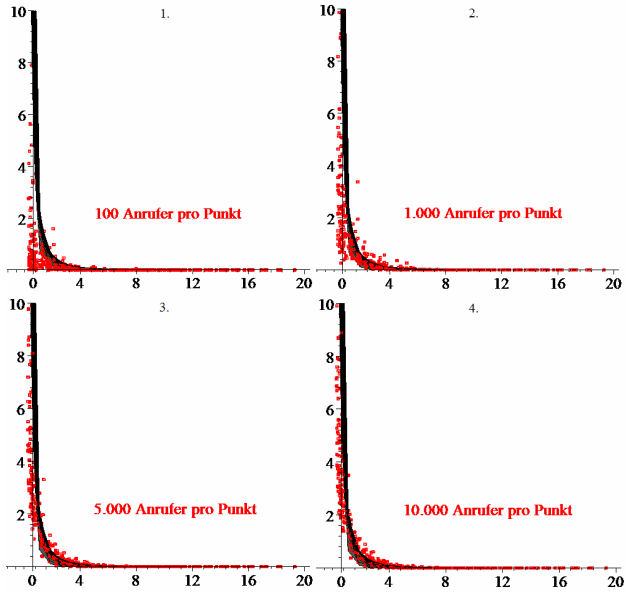


Abb. 8. Genauigkeit der Simulationen für die mittlere Wartezeit: jeweils die Seitenansicht der Abbildung 6 (rechts) mit 1.) 100 Anrufer pro Punkt, 2.) 1.000 Anrufer pro Punkt, 3.) 5.000 Anrufer pro Punkt und 4.) 10.000 Anrufer pro Punkt; die Punkte „ziehen von unten" immer näher an die tatsächliche analytische Lösung, da das Einschwingen an Bedeutung verliert.

Werden nur wenige Anrufer simuliert, wie z. B. 100 Anrufer (Abbildung 8 1.), so ist schnell ersichtlich, dass bei 20 Call Center Agenten für die ersten 20 Anrufer keine Wartezeiten und für die folgenden kaum Wartezeiten entstehen, da die Agenten den Anstrom an Kunden sehr leicht bewältigen können. Die Simulation für die mittlere Wartezeit befindet sich noch in der Einschwingphase (Anlaufphase) in den stationären Zustand. Wird die Anzahl der simulierten Anrufe schrittweise von 100 auf 1.000, 5.000 und 10.000 erhöht, so hat die Anlaufphase immer weniger Auswirkung auf die Simulationsdaten (vgl. Abbildung 8 2., 3. und 4.) und das System stabilisiert sich. Der stationäre Zustand der einzelnen Simulationspunkte ist abhängig von der Ankunfts- und der Bedienrate. Während bei einem Agenten nur ca. 1.000 Anrufer simuliert werden müssen, sind es bei 20 schon über 5.000 Anrufer um nahezu den stationären Zustand zu erreichen[14]. Im Folgenden werden daher nur noch die Simulationsdaten mit 5.000 bzw. 10.000 simulierten Anrufern pro Punkt für das Training der neuronalen Netze benutzt. Es sei hier schon darauf hingewiesen, dass sich das System nur annähernd im stationären Zustand befinden muss, da die neuronalen Netze ein Rauschen in den Simulationsdaten sehr gut ausgleichen können. Dennoch sollten sich die Simulationsdaten sehr nah an der analytischen Lösung befinden. Besonders wenn der Bereich betrachtet wird, wo die Ankunftsrate gegen die Bedienrate geht und somit die mittlere Wartezeit sprunghaft ansteigt und die Simulationsdaten nur noch unterhalb der analytischen Lösung sind (vgl. Abbildung 8 und Abbildung 6 rechts). Aber in einem Call Center sind aus Servicegründen nur geringe Wartezeiten der Kunden erwünscht, so dass eigentlich nur der Bereich analysiert werden muss, wo die Wartezeiten nahezu Null sind bzw. nur leicht ansteigen[15]. In diesem Bereich liegt die analytische Lösung bei 5.000 bzw. 10.000 simulierten Anrufern direkt in den Simulationsdaten (vgl. Abbildung 8 3. und 4. jeweils der untere Bereich bis zu 2 Minuten Wartezeit).

[14] Es gibt keine exakten statistischen Verfahren zur Bestimmung der Anlaufphase, nur heuristische Ansätze wie die Regeln nach Conway, bzw. nach Tocher oder nach Morse, wobei letztere eine Abschätzung liefert: Anlaufzeit > 3 · Ankunftsrate / (Ankunftsrate – Bedienrate)^2, vgl. Page (1991) und Ripley (1987).

[15] In der Praxis beträgt die maximale Zeit, die ein Kunde warten darf, meist nur wenige Sekunden. Wir betrachten hier dennoch den Bereich weit über zwei Minuten Wartezeit, dementsprechend müssen 5.000 bis 10.000 Anrufer simuliert werden, obwohl für den Praxisfall Call Center weniger gereicht hätten.

6 Approximation von Kennzahlen für Warteschlangensysteme

Wesentliche Vorteile der Approximation gegenüber der einfachen diskreten Simulation von Kennzahlen sind,

- dass eine kontinuierliche Funktion zur Kostenminimierung generiert wird, und
- dass die approximierte Funktion eine bessere Annäherung an die analytische Lösung aufweist als die Simulationsdaten.

Letzteres ist dadurch begründet, dass die Simulationsdaten immer ein Rauschen aufweisen und die approximierte Funktion in diesen Daten liegt. Da auch stärkere Schwankungen der verwendeten Musterdatensätze durch das neuronale Netz wieder ausgeglichen werden, ist die Simulation, die der Approximation durch den Neurosimulator FAUN vorangestellt ist, zeitlich wesentlich weniger aufwendig, als wenn die gewünschte Kennzahl nur alleine durch Simulation bestimmt werden soll. Wichtig ist jedoch, dass die zugrunde liegende Simulation annähernd den stationären Zustand erreicht und somit hinreichend nahe der analytischen Lösung ist (vgl. Kapitel 5). Da der weitere Arbeitsschritt durch die Approximation mit FAUN nur wenige Sekunden beträgt, entsteht hierdurch kein wesentlicher Nachteil.

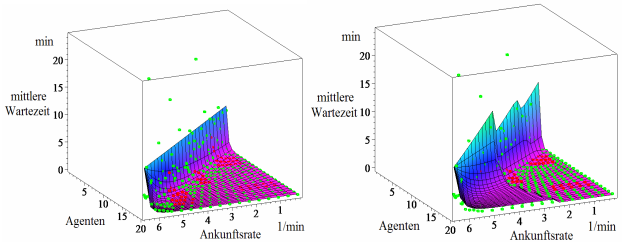


Abb. 9. Das Neuronale Netz mit einem verdeckten Neuron (links) weist einen höheren Trainingsfehler auf als das neuronale Netz mit drei inneren Neuronen (rechts), dennoch ist das rechte Netz unbrauchbar für die Kennzahlenbestimmung, da es in den Daten oszilliert.

6.1 Approximation mit FAUN 1.0

Nach der Aufteilung der 349 Simulationsdaten[16] in $n_t = 285$ Trainings- und $n_v = 64$ Validierungsdaten[17] und der Equilibrierung und Skalierung aller Muster stellt sich beim Training der neuronalen Netze schnell heraus, dass dreilagige Perceptrons ohne Shortcuts mit nur einem inneren Neuron in der verdeckten Schicht die besten Resultate für die Approximation der mittleren Wartezeit liefern[18] (vgl. Abbildung 9). Dabei wurden Topologien untersucht, bei denen die innere Neuronenanzahl n_2 von 1 bis 10 variierte. Gemäß (7) wurde zu den einzelnen Topologien der Trainingsfehler ε_t und Validierungsfehler ε_v bestimmt.

Neuronale Netze mit einer höheren Anzahl an inneren Neuronen weisen zwar einen geringeren Trainings- und Validierungsfehler auf (vgl. Tabelle 1), sind aber zur Kennzahlenbestimmung unbrauchbar, da sie nicht mehr eine „glatte Fläche“ aufweisen, sondern „wellig“ sind (vgl. Abbildung 9 und 10). Dies ist auch schon bei zwei inneren Neuronen der Fall.

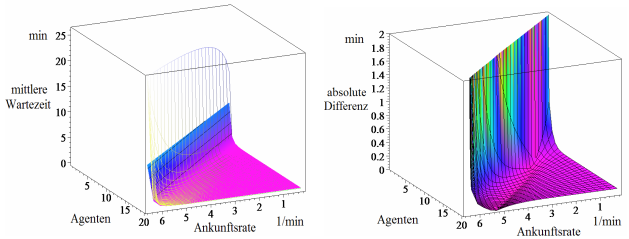


Abb. 10. Links: Vergleich der analytischen Lösung für die mittlere Wartezeit (Gitternetz) mit dem neuronalen Netz (Fläche) bei 10.000 simulierten Anrufern pro Punkt. Rechts: Absolute Differenz der beiden Lösungen in Minuten.

Neuronale Netze mit mehreren inneren Neuronen neigen dazu zwischen den Daten zu oszillieren um diese auswendig zu lernen. Diese Oszillation ist aber in vielen Praxisanwendungen, so auch hier, nicht erwünscht und so liefern neuronale Netze mit nur wenigen inneren Neuronen trotz eines höheren Trainingsfehlers bessere Ergebnisse. Daher ist eine graphische Analyse empfehlenswert bzw. in einer mathematischen Analyse muss der

[16] Vgl. Abbildungen 5 und 9.

[17] Dabei sollten Trainings- und Validierungsdaten so gewählt werden, dass $1 \le n_t/n_v \le 9$ gilt (vgl. Breitner (2003)).

[18] Dies wurde auch schon für das M/M/1 gezeigt (Barthel 2003).

Krümmungstensor möglichst klein sein (vgl. Breitner 2003). Dies zeigt auch der graphische Vergleich mit der analytischen Lösung für die zu erwartende Wartezeit in der Schlange aus (4) in Abbildung 10 (links). Die absolute Differenz der beiden Lösungen (Abbildung 10 rechts) ist fast über den ganzen Bereich nahezu immer Null. Nur in dem Bereich, wo die Ankunftsrate gegen die kombinierte Bedienrate geht, steigt die absolute Differenz sprungartig an, da hier die analytische Lösung gegen unendlich divergiert. Anhand dieser graphischen Betrachtung ist schon zu erkennen, wie gut das neuronale Netz diese Kennzahl approximiert. Die entsprechende mathematische Analyse bezüglich der Abweichungen der beiden Verfahren wird in Tabelle 2 dargestellt.

Tabelle 1. Trainings- und Validierungsfehler der besten Approximationsfunktionen

Topologie	1 inneres Neuron	2 innere Neuronen	3 innere Neuronen	5 innere Neuronen	10 innere Neuronen
ε_t*	2,64	2,56	2,49	1,95	1,91
prozentualer Fehler	7,3 %	7,2 %	7,1 %	6,3 %	6,2 %
ε_v*	0,64	0,74	0,55	0,47	0,53
Rechenzeit in sec.	6,4	10,7	14,9	23,0	43,5

6.2 Qualität der Approximation

Um die Approximation der mittleren Wartezeit in der Warteschleife mit der analytischen Lösung und den tatsächlichen Simulationspunkten vergleichen zu können, ist zu beachten, dass die Inputwerte für das neuronale Netz skaliert eingehen, während die Ausgabe zurückskaliert werden muss, so dass die Wartezeit wieder in Minuten abzulesen ist.

In der Tabelle 2 werden die drei Verfahren Approximation, Simulation und analytische Lösung der mittleren Wartezeit in der Warteschleife für das entsprechende Beispiel des Inbound-Call-Centers mit maximal 20 Agenten verglichen. Dazu werden für drei verschiedene Bereiche des Lösungsraumes, aufgespannt durch die Ankunftsrate, die kombinierte Bedienrate und die zugehörige mittlere Wartezeit, die minimale, maximale

und durchschnittliche Abweichung der drei Verfahren untereinander in Minuten bestimmt.

Die approximierte Lösung auf Basis von neuronalen Netzen hat bis auf einen Wert immer eine geringere durchschnittliche Abweichung zu der analytischen Lösung als die beiden Simulationen einmal mit 5.000 Anrufern pro Punkt und einmal mit 10.000 Anrufern pro Punkt (vgl. Tabelle 2)[19].

Wird der gesamte Bereich (jeweils die ersten drei Zeilen der Tabelle 2) betrachtet, so treten hier die größten maximalen Abweichungen auf. Dies ist dadurch begründet, dass, wenn die Ankunftsrate λ sich der kombinierten Bearbeitungsrate $c\mu$ annähert, die analytische Lösung sehr schnell gegen unendlich geht und auch die Simulationspunkte in diesem Bereich größere Schwankungen aufweisen. Dennoch beträgt die durchschnittliche Abweichung des neuronalen Netzes zur exakten Lösung jeweils nur etwas mehr als eine Minute, da auch der Bereich betrachtet wird, wo alle drei Verfahren eine mittlere Wartezeit von nahezu Null haben.

Tabelle 2. Vergleich des besten künstlichen neuronalen Netzes (NN) mit der analytischen Lösung (Ana.) und den Simulationsdaten (Simu.) (alle Werte in Minuten angegeben)

Anzahl Anrufer		Ana. vs. NN		NN vs. Simu.		Ana. vs. Simu.	
		5.000	**10.000**	**5.000**	**10.000**	**5.000**	**10.000**
$E(W_q)$ gesamt 349 Pkt	min Abw.	0,0003	0,0007	0,0002	0,0001	$2\cdot10^{-21}$	$2\cdot10^{-21}$
	max Abw.	19,04	17,25	9,37	16,76	22,87	22,15
	Ø Abw.	**1,45**	**1,32**	0,51	0,67	1,65	1,45
$E(W_q) \leq 5$ 280 Pkt	min Abw.	0,0003	0,0007	0,0002	0,0001	$2\cdot10^{-21}$	$2\cdot10^{-21}$
	max Abw.	1,32	1,33	3,09	2,94	3,64	2,89
	Ø Abw.	**0,12**	**0,16**	0,19	0,23	0,20	0,16
$0,1 \leq E(W_q) \leq 5$ 122 Pkt	min Abw.	0,0003	0,0007	0,003	0,002	0,0005	0,01
	max Abw.	1,32	1,33	3,09	2,94	3,64	2,89
	Ø Abw.	**0.24**	**0.26**	0,39	0,42	0,47	0,37

[19] Ausnahme bildet der zweite Bereich bei 10.000 simulierten Anrufern, da sind beide Werte gleich 0,16.

Dementsprechend werden noch zwei zusätzliche Bereiche analysiert. Zum einen wird der Lösungsraum betrachtet für den die mittlere Wartezeit nicht größer als fünf Minuten ist ($E(W_q) \leq 5$), da hier angenommen wird, dass dies für die Kunden des Inbound-Call-Centers eine zumutbarer Wartezeit ist[20]. Zum anderen wird der Lösungsraum analysiert für den zusätzlich der Bereich nicht betrachtet wird, wo alle drei Verfahren fast Null sind ($0,1 \leq E(W_q) \leq 5$). Es ist ersichtlich, dass das neuronale Netz, welches durch nur 5.000 Anrufer pro Simulationspunkt generiert wurde, geringere Werte aufweist als das beste neuronale Netz mit 10.000 Anrufern pro Punkt. Die durchschnittliche Abweichung beträgt dann nur noch ungefähr sechs (für $E(W_q) \leq 5$) bzw. 12 Sekunden (für $0,1 \leq E(W_q) \leq 5$) und die maximale Abweichung etwas mehr als eine Minute, vgl. Tabelle 2. Es ist daher anzunehmen, dass für diese eingeschränkten praxisrelevanten Bereiche jedoch 5.000 simulierte Anrufer zur Generierung der neuronalen Netze ausreichen, obwohl die approximierte Lösung, generiert durch 10.000 Anrufer pro Punkt, insgesamt eine bessere durchschnittliche Abweichung aufweist. Das beste neuronale Netz weicht auf einem Intervall von null bis fünf Minuten für die mittlere Wartezeit nur im Durchschnitt sechs Sekunden von der analytischen Lösung ab.

6.3 Auswertung des Inbound-Call-Centers

Neben der durchschnittlichen Wartezeit der Kunden in der Warteschleife ist für einen Call-Center-Manager noch der Auslastungsgrad bzw. Servicegrad ρ seiner Agenten entscheidend. Wenngleich auch der Auslastungsgrad hier durch (1) sehr einfach bestimmt werden kann, wurde er mit simuliert und dann mit FAUN approximiert, da ρ bei weit aus schwierigeren Problemen nicht mehr so leicht zu bestimmen ist (z. B. sind nicht alle Agenten identisch und haben alle die gleiche Bedienrate μ).

Während die Ankunftsrate tageszeitabhängig und exogen ist (vgl. Abbildung 1), d. h. nicht beeinflussbar vom Call-Center-Manager[21], ist die Anzahl an Agenten dagegen endogen, also steuerbar. In Abbildung 11 ist zu erkennen, dass, wenn die Ankunftsrate gegen die kombinierte Bedienrate geht, die durchschnittliche Wartezeit (links) lange Zeit Null bleibt, dagegen aber der Auslastungsgrad (rechts) ständig steigt. Ein Call-Center-Manager ist also bei vorgegebener Ankunftsrate bestrebt, einen möglichst

[20] Oft liegt diese obere Grenze in der Praxis doch weit niedriger.

[21] Die Warteschleifen in Call Centern haben eine vom System bzw. auch vom Call-Center-Manager vorgegebene bzw. einstellbare Kapazität, so dass Kunden bei voller Warteschleife abgewiesen werden, und somit kann indirekt Einfluss genommen werden.

hohen zumutbaren Auslastungsgrad seiner Agenten gegenüber möglichst geringen Wartezeiten der Kunden zu erreichen. Die approximierten Lösungen dieser beiden Kennzahlen sind also nur dann sinnvoll einsetzbar, wenn ein hinreichend genaues Prognoseverfahren für die Ankunftsrate der nächsten Stunden bzw. Tage zur Verfügung steht.

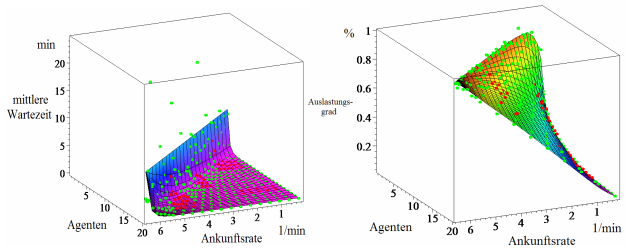


Abb. 11. Approximierte durchschnittliche Wartezeit der Kunden in der Warteschleife (links) und der dazugehörige approximierte Auslastungsgrad (rechts) mit den jeweiligen Simulationspunkten.

Soll zum Beispiel die mittlere Wartezeit eines Kunden nur eine Minute betragen und die prognostizierte Ankunftsrate für die nächsten Zeitintervalle fünf Kunden pro Minute beträgt, so liefert die approximierte Lösung, dass 16,32 Agenten, also 17, eingesetzt werden müssen. Dabei beträgt der approximierte Auslastungsgrad der 17 Agenten 87,85%. Die analytische Lösung für die mittlere Wartezeit in der Schleife liefert einen Wert von 16,49 Agenten, also auch 17, und der Auslastungsgrad der 17 Agenten beträgt bei der analytischen Lösung 88,23%.

7 Fazit und Ausblick

Der Neurosimulator FAUN bietet eine Möglichkeit, für alle Warteschlangensysteme eine approximierte, explizite Lösung für deren Kennzahlen zu generieren. Dieser Aufsatz zeigt anhand des Standardmodells M/M/*c* wie dies möglich wird und welche Güte die approximierte Lösung im Gegensatz zur – hier ermittelbaren – analytischen Lösung besitzt. Die so gewonnenen Erkenntnisse können auf Modelle ohne analytische Lösung übertragen werden, die bisher nur mit Simulationen gelöst werden können.

Vorteile der Approximation von Warteschlangenkennzahlen bei schwierigen Warteschlangenproblemen gegenüber der Analyse durch diskrete Simulationen sind,

- dass eine analytische Funktion zur Personaleinsatzplanung und Kostenminimierung generiert wird, die extrem schnell auswertbar ist, und
- dass das unvermeidliche Rauschen in den Simulationsdaten geglättet wird, d. h. die Kennzahlen genauer verfügbar sind bzw. deutlich weniger Simulationen nötig sind.

Es brauchen nicht besonders viele Punkte zum Training der neuronalen Netze simuliert werden gegenüber einer „flächendeckenden“ Auswertung mit einer Simulation. Dies ist ein erheblicher Zeitvorteil, da der zusätzliche Schritt des FAUN-Trainings i. d. R. nur wenige Sekunden dauert.

Ein weiterer Vorteil ist, dass bei der Simulation der Muster für das Training mit FAUN unterschiedlichste Verteilungen für die Ankunfts- und Bedienrate, so wie sie in der Praxis tatsächlich vorkommen, eingesetzt werden können. Beispielsweise kann so anhand des realen Anruferaufkommens in einem Call Center die tatsächliche Verteilung über einen längeren Zeitraum bestimmt und für die Simulation verwendet werden. Analog kann mit dem Bedienprozess verfahren werden. Aus realen Daten können dann Simulationspunkte für das Training der künstlichen neuronalen Netze generiert werden, um so die Abläufe in einem Call Center durch realistischere Bestimmung der Warteschlangenkennzahlen wesentlich praxisnäher abzubilden. Es muss also nicht das M/M/*c*-Modell mit all seinen Einschränkungen als Grundlage für die Mustergenerierung dienen. Dieses aus der Praxis gewonnene Datenmaterial kann durchaus verrauscht sein, da neuronale Netze mit wenigen inneren Neuronen sich „in die Daten legen“ und so oft ein gleichmäßiges, oft weißes Rauschen ausgleichen.

Bevor jedoch schwierigere Warteschlangenprobleme ohne analytische Lösung anhand neuronaler Netze analysiert werden, muss weiter untersucht werden, ab wann die Simulationspunkte in Abhängigkeit von der Ankunfts- und Bedienrate nahezu ihren stationären Zustand erreichen. Wichtig ist, dass der jeweilige Einschwingvorgang der Simulationen nur wenig mitgelernt wird. Ein Ansatz zur besseren Generierung der Simulationsdaten wäre zum Beispiel, dass die Auswertung der Wartezeiten bzw. die Bestimmung der Kennzahlen erst nach einer gewissen Anzahl von ankommenden Kunden anfängt, um so den Vorgang des Einschwingens abzuschneiden.

Literatur

Barthel A (2003) Effiziente Approximation von Kenngrößen für Warteschlangen mit dem Neurosimulator FAUN 1.0. Diplomarbeit am Institut für Wirtschaftswissenschaft der Universität Hannover, Königsworther Platz 1, D-30167 Hannover

Bolch G (1989) Leistungsbewertung von Rechensystemen mittels analytischer Warteschlangenmodelle. Teubner, Stuttgart

Box GEP, Jenkins G M, Reinsel G C (1994) Time Series Analysis: Forecasting and Control, 3. Aufl. Prentice Hall, New Jersey

Breitner MH (2003) Nichtlineare, multivariate Approximation mit Perzeptrons und anderen Funktionen auf verschiedenen Hochleistungsrechnern. Akademische Verlagsgesellschaft Aka GmbH, Berlin

Call Center-Benchmark Kooperation (2004) Kooperationsprojekt: Purdue University, Universität Hamburg, Initiator der profiTel MANAGEMENT CONSULTING. http://www.callcenter-benchmark.de/index3.html. Letzter Abruf: 10.10.2004

Datamonitor (2004) Datamonitor. http://www.datamonitor.com. Letzter Abruf 21.05.2004

Domschke W, Drexl A (2002) Einführung in Operations Research, 5. Aufl. Springer, Berlin

Helber S, Stolletz R (2004) Call Center Management in der Praxis: Strukturen und Prozesse betriebswirtschaftlich optimieren. Springe, Berlin

Henn H, Kruse JP, Strawe OV (1998) Handbuch Call Center Management (2. Aufl.). Telepublic Verlag, Hannover

Hillier FS, Lieberman GJ (1997) Operations Research, 5. Aufl. Oldenbourg, München Wien

Kestling V (2004) Das Call Center-Jahr 2003 – Rückblick und Ausblick. Markt & Trends CallCenter, CRM, IT/TK, Telesales und –services, Presse Mitteilung

Meyer M, Hansen K (1996) Planungsverfahren des Operations Research (4. Aufl.). Vahlen, München

Page B (1991) Diskrete Simulation. Springer, Berlin

Ripley BD (1987) Stochastic Simulation. John Wiley & Sons, New York

Schassberger R (1973) Warteschlangen. Springer, Berlin

Siegert HJ (1991) Simulation zeitdiskreter Systeme. Oldenbourg, München Wien

Zimmermann W (1997) Operations Research: quantitative Methoden zur Entscheidungsvorbereitung (8. Aufl.). Oldenbourg, München Wien

Seminararbeit am Institut für Volkswirtschaftslehre

Lehrstuhl Geld und Internationale Finanzwirtschaft

Universität Hannover

Wirtschaftswissenschaftliche Fakultät

Wechselkursprognosen mit Hilfe von neuronalen Netzen am Beispiel des Thai Baht-US-Dollar Wechselkurses

Dipl.-Math. Frank Köller

Matr.-Nr. 2199868

Hugenottenstrasse 19

31785 Hameln

Tel.: 05151 / 3299

E-Mail-Adresse: koeller@iwi.uni-hannover.de

Betreuerin: Dipl.-Ök. Daniela Beckmann

Rahmenthema: Banken- und Währungsfragen in Schwellenländern

SS 2005

1 Einleitung

Neuronale Netze werden zunehmend für Anwendungsgebiete in der Finanzwirtschaft eingesetzt. Eine Anwendungsmöglichkeit besteht in der Prognose von Wechselkursen. In dieser Arbeit soll anhand der thailändischen Währungen Thai Baht und dem US-Dollar verdeutlicht werden, wie Wechselkurprognose mit neuronalen Netzen erstellt werden können. Wechselkursprognosen sind von besonderer Bedeutung, da der Devisenmarkt täglich große Umsätze abwickelt. Einzelne Transaktionen können das Volumen von 1 Mrd. US-Dollar erreichen oder überschreiten. Bei derartigen Summen können selbst kleinste Schwankungen im Kurs einen großen Betrag ausmachen.
Im Gegensatz zu reinen technischen Analysen, die sich nur auf den Wechselkursverlauf beziehen, wird hier versucht kurzfristige und mittelfristige Prognosen unter Einbezug einer breiten Fundamental-Datenbasis zu erstellen. Dementsprechend gehen nicht nur der Kursverlauf, sondern auch die jeweiligen Inflationsraten, in- und ausländische Zinsen, Volumen von Exporten und Importen und die Währungsreserven Thailands verstärkt durch verschiedene technische Indikatoren in das Prognosemodel ein. Ein Zusammenhang zwischen diesen Einflussgrößen und den Wechselkursschwankungen wird in Kapitel 2 gegeben.
Nach einer kurzen Einführung in die künstliche Intelligenz in Kapitel 3 wird in Kapitel 4 erläutert, wie kurzfristige und mittelfristige Prognosen[1] mit dem Neurosimulator FAUN[2] erstellt werden und welche Güte diese Prognosen haben. Dabei wird insbesondere untersucht, inwieweit die Asienkrise (1997) Einfluss auf das Training der neuronalen Netze hat, bzw. ob diese Krise mit neuronalen Netzen vorhergesagt werden hätte können.

2 Ausgewählte Wechselkurstheorien im Überblick[3]

Fundamentale Wechselkurstheorien basieren in der Regel auf den in diesem Abschnitt beschriebenen Ansätzen der Kaufkraftparitätentheorie und der Zinsparitätentheorie. Im Rahmen des klassischen monetären Ansatzes der Wechselkursbestimmungen ist die Gültigkeit der Kaufkraftparitätentheorie eine der Hauptprämissen. Bei den neueren monetären Ansätzen wird zudem noch von der Gültigkeit der ungesicherten Zinsparität ausgegangen. Beiden Ansätzen ist gemein, dass Veränderungen des Preisniveaus bei permanent angenommener Gültigkeit der Kaufkraftparitätentheorie auch Auswirkungen auf den gleichgewichtigen Wechselkurs haben.

2.1 Kaufkraftparitätentheorie

Ziel der Kaufkraftparitätentheorie ist die langfristige Entwicklung des Wechselkurses zu erklären. Dabei wird allgemein zwischen der absoluten und der komparativen Form der Kaufkraftparität (KKP) unterschieden.

[1] Der Prognosehorizont beträgt hier ein und zwei Monate, da alle Einflussgrößen nur auf Monatsbasis vorhanden sind.

[2] „**F**ast **A**pproximation with **U**niversal **N**eural Networks“. Neurosimulator bezieht sich nicht auf die Simulation von Wechselkursprognosen, sondern auf die komfortable, GUI-unterstützte Simulation gehirnanaloger Vorgänge, die als Training bzw. Lernen von künstlichen neuronalen Netzen bekannt sind (Breitner 2003).

[3] Es wird kein Anspruch auf eine vollständige Überblicksdarstellung erhoben, sonder vielmehr sollen die später verwendeten Einflussgrößen für das Training der neuronalen Netze im Bezug auf den Wechselkurs erklärt werden.

Wird ein vollkommener Weltmarkt vorausgesetzt, so impliziert dies zum einen, dass die Wirtschaftssubjekte die im In- und Ausland produzierten Güter als vollständig substitutiv ansehen (vgl. Dieckheuer (2001), S.285). Somit handelt es sich um perfekt homogene Güter, bei denen keine Präferenzen sachlicher, räumlicher oder zeitlicher Art seitens der Anbieter und Nachfrager bestehen. Zum anderen herrscht national sowie international vollständige Markttransparenz und es existieren keine preisverzerrenden Handelsbeziehungen z.B. in Form von Steuern, Zöllen oder Kontingenten. Zudem besteht ein flexibles Preissystem bei Existenz vieler Anbieter und Nachfrager. Bei Vernachlässigung von Transportkosten und bei ausschließlicher Betrachtung von handelbaren Gütern muss das Gesetz des einheitlichen Preises (law of one price) gelten. Demnach muss unter Berücksichtigung des Wechselkurses (w) der Geldwert der Währungen in beiden Ländern übereinstimmen, bzw. die Kaufkraftparität ist erfüllt, wenn in beiden Ländern für einen bestimmten, in die jeweilige Landeswährung umgerechneten Geldbetrag die gleiche Gütermenge erworben werden kann (Absolute Form der Kaufkraftparität):

$$p = p_a\, w \tag{1}$$

Unter diesen idealen Bedingungen führen Arbitragegeschäfte zwar bei homogenen international gehandelten Gütern zu einer Übereinstimmung des Preises. Es ist aber zu berücksichtigen, dass tatsächlich nur ein Teil des internationalen Handels auf homogene standardisierte Güter wie Rohstoffe entfällt, während ein großer Teil aus heterogenen Gütern (z.B. Maschinen) besteht. Zudem wirken Transportkosten und Handelsbeschränkungen dem Gesetz des einheitlichen Preises entgegen und heterogene Güter sind gegeneinander nur unvollkommen substituierbar. Daher besteht keine ökonomische Notwendigkeit zu übereinstimmenden Preisen. Ferner können auch noch Waren und Dienstleistungen berücksichtigt werden, die typischerweise nur national gehandelt werden. Es wird deutlich, dass kein Angleichen aller Preise erfolgen kann, die in das Preisniveau der jeweiligen Länder eingehen.
Den Bedenken gegen die absolute Form der Kaufkraftparität trägt die abgeschwächte Formulierung der komparativen Form Rechnung:

$$p^x = \gamma w p_a{}^x\,, \tag{2}$$

wobei p^x und $p_a{}^x$ die Preisindices für die Lebenshaltung im In- und Ausland sind (vgl. Jarchow und Rühmann (2000)). Demnach kann, wenn $\gamma \neq 1$ ist, das heimische Preisniveau von der Kaufkraftparität in der absoluten Form abweichen. Es wird aber unterstellt, dass sich γ im Zeitablauf nicht verändert und damit bleibt auch der reale Wechselkurs ($w p_a{}^x / p^x$), der nach Gleichung (2) dem Kehrwert von γ entspricht, im Zeitablauf konstant. Die Kaufkraftparität in der komparativen Form setzt voraus, dass sich die Preise nationaler und international gehandelter Güter proportional verändern (vgl. Jarchow (2000) S.268).

2.2 Zinsparitätentheorie

Das Zinsparitätentheorem ist ein Modell, um kurzfristige Reaktionen des Wechselkurses zu erklären. Im Gegensatz zur Kaufkraftparitätentheorie, bei dem ein vollkommener Weltmarkt Voraussetzung ist, wird hier ein vollkommener Kapitalmarkt unter Betrachtung von homogenen Wertpapieren zu Grunde gelegt. Dies impliziert, dass die Finanzaktiva für die Wirtschaftssubjekte bei vollkommener Kapitalmobilität perfekte Substitute sind. Bei Wirksamkeit der Arbitrage kann es zu keinem Unterschied zwischen den Renditen in- und ausländischer Finanzaktiva kommen.
In diesem Modellrahmen wird ein Anleger betrachtet, der als Gegengeschäft zum Kauf der Fremdwährungsguthaben am Kassamarkt, bereits zu Beginn der Periode den Verkauf per Termin vereinbart und somit das Wechselkursrisiko bei einer Fremdwährungsanlage ausschaltet (Kurssicherung am Terminmarkt). Die Verknüpfung der Zinsarbitrage mit der Terminspekulation liefert schließlich einen einfachen Ansatz zur kurzfristigen Wechselkursbestimmung. Da das Wechselkursrisiko durch das Termingeschäft ausgeschaltet ist und von

Transaktionskosten abgesehen wird, sorgt der Marktmechanismus dafür, dass Ertragsunterschiede zwischen In- und Auslandsanlagen beseitigt werden. Da diese Anpassung ohne nennenswerte Verzögerung erfolgt, wird dies auch als gesicherte Zinsparität bezeichnet:

$$w = \frac{1+i_a}{1+i}\, w_T\,. \tag{3}$$

Damit wird deutlich, dass Kassa- und Terminkurs (w und w_T) bei gegebenen in- und ausländischen Zinssätzen (i und i_a) in einem proportionalen Verhältnis zueinander stehen. Wird zusätzlich noch die Tätigkeit der Spekulanten betrachtet, so wird sich bei Risikoneutralität der Terminkurs vollständig an den erwarteten Kassakurs (w^{erw}) anpassen:

$$w_T = w^{erw}. \tag{4}$$

Demzufolge beeinflussen der künftig erwartete Kassakurs sowie die Zinssätze in In- und Ausland als exogene Größen den laufenden Kassakurs:

$$w = \frac{1+i_a}{1+i}\, w^{erw}\,. \tag{5}$$

Diese Gleichung wird auch als ungesicherte Zinsparität bezeichnet.
Vorausgesetzt wird hier, dass die auf die Wechselkurserwartungen einwirkenden Einflüsse die Zinssätze unberührt lassen. In der Realität fallen zudem zusätzlich Transaktionskosten an und nicht alle Terminspekulanten verhalten sich risikoneutral. Störungen, wie geldpolitische Maßnahmen, die Auswirkungen auf die Zinssätze haben, werden im Regelfall auch die Wechselkurserwartungen beeinflussen. Änderungen der Zinssätze lösen internationale Kapitalbewegungen aus: wird z.B. durch geldpolitische Maßnahmen im Inland die Zinssätze i gesenkt oder durch geldpolitische Maßnahmen im Ausland die Zinssätze i_a erhöht führt dies zu einem Wechsel von kursgesicherten Kapitalimporten zu kursgesicherten Kapitalexporten und damit tritt eine Veränderung des Wechselkurses ein (gesunkener Termin- und erhöhter Kassakurs). Empirische Untersuchungen kommen aber zu dem Ergebnis, dass unter Berücksichtigung von Transaktionskosten die gesicherte Zinsparität mit Einschränkungen auch zwischen nationalen Währungen gilt (vgl. Käsmeier (1984)). Die Einschränkungen resultieren insbesondere daraus, dass ein politisches Risiko, z.B. in Form von Kapitalverkehrskontrollen, existiert. Dies kann dazu führen, dass Geschäfte unterbleiben, obwohl sie bei den herrschenden Zinssätzen und den geltenden Devisenkursen einen Gewinn versprechen. Insgesamt gesehen kann aber die gesicherte Zinsparität als eine gute Annäherung an die Wirklichkeit betrachtet werden (vgl. MacDonald und Taylor (1992) bzw. Sarno und Taylor (2002)).

2.3 Monetärer Ansatz zur Wechselkursbestimmung

Der klassische monetäre Ansatz kann als Erweiterung der Kaufkraftparitätentheorie angesehen werden, da die Geldmenge explizit über den quantitätstheoretischen Zusammenhang als Bestimmungsfaktor des Preisniveaus Berücksichtigung findet. Bei diesem Ansatz wird zum einen die permanente Gültigkeit der Kaufkraftparität unterstellt und zum anderen die Gültigkeit der Quantitätstheorie (vgl. Pentecost (1993) S.20f. und Jarchow (2003)). Zudem soll auch in beiden Länder Vollbeschäftigung herrschen und daher werden in- und ausländisches reales Volkseinkommen als exogen gegeben angesehen. Zinsen, Inflations- und Wechselkurserwartungen werden von der Analyse ausgeschlossen. Beide Länder produzieren und konsumieren zudem das gleiche handelbare Gut oder Güterbündel. Die nominalen in- und ausländischen Geldmengen sollen ebenfalls exogen vorgegeben sein. Bei Berücksichtigung der Vollbeschäftigungsniveaus (Y^r*) für das jeweilige Sozialprodukt[4], lautet dann die in- und ausländischen Geldnachfragefunktionen:

[4] Das reale Sozialprodukt (Y^r)erreicht im langfristigen Gleichgewicht sein Vollbeschäftigungsniveau (Y^r*)

$$M = kp^x\, Y^{r*}, \tag{6}$$

$$M_a = k_a p_a{}^x\, Y_a{}^{r*}, \tag{7}$$

wobei k und k_a in- und ausländische Kassenhaltungskoeffizienten sind. Werden diese beiden Beziehungen nach dem Preisniveau aufgelöst und die Kaufkraftparität in ihrer absoluten Form zu Grunde gelegt ergibt sich für den Wechselkurs

$$w = \frac{p^x}{p_a{}^x} = \frac{M}{M_a}\frac{k_a Y_a{}^{r*}}{kY^{r*}}. \tag{8}$$

Wie aus Gleichung (8) hervorgeht, liegt der Wechselkurs umso niedriger, je geringer das inländische Geldangebot (M) im Verhältnis zum ausländischen Geldangebot (M_a) ist. Steigt beispielsweise das inländische Geldangebot bei gegebenen Realeinkommen in In- und Ausland unterproportional zum ausländischen Geldangebot an, dann geht das Preisniveau des Inlands im Verhältnis zum Preisniveau des Auslands zurück. Gemäß der Kaufkraftparitätentheorie ergibt sich daraufhin ein Rückgang des Wechselkurses, d.h. eine Aufwertung der heimischen Währung.
Im Unterschied zu dem klassischen monetären Ansatz wird in dem Grundmodell des neueren monetären Ansatzes der internationale Kapitalverkehr mit in die Analyse einbezogen und von der Existenz eines in- und ausländischen Wertpapiermarktes ausgegangen. Da annahmegemäß in- und ausländische Wertpapiere perfekte Substitute sind und keine Kapitalverkehrsbeschränkungen existent sein sollen, impliziert dies bei Vernachlässigung von Transaktionskosten die permanente Gültigkeit der ungesicherten Zinsparität. Gleichung (8) ändert sich unter Berücksichtigung der Kaufkraftparität in ihrer Komparativen Form zu

$$w = \frac{1}{\gamma}\frac{M}{M_a}\frac{L_a{}^r\left(i_a, Y_a{}^{r*}\right)}{L^r\left(i, Y^{r*}\right)} \tag{9}$$

wobei L^r und $L_a{}^r$ der realen Geldnachfrage im In- und Ausland entsprechen.

2.4 Keynesianische Ansätze zur Wechselkursbestimmung

Im Unterschied zu den monetären Ansätzen findet bei den handelsbilanzorientierten Ansätzen der Wechselkursbestimmung die Kaufkraftparitätentheorie aufgrund eines angenommenen kurzfristig fixen Preisniveaus keine Berücksichtigung. Vielmehr wird die Höhe des Wechselkurses ausschließlich durch das Devisenangebot und die Devisennachfrage bestimmt, welches aus Ex- und Importen von Gütern resultiert. Im Gegensatz zu den monetären Ansätzen, bei denen Bestandsgrößen im Vordergrund stehen handelt es sich bei diesem Ansatz um eine reine Stromgrößenbetrachtung. Die Gleichung der Handelsbilanz (HB), die in diesem Fall identisch mit der Zahlungsbilanz ist, da Kapitalverkehrs-, Dienstleistungs- und Übertragsbilanz vernachlässigt werden sollen, lautet:

$$HB = X - w\,J = 0 \tag{10}$$

wobei X der Exportwert in inländischer Währung und J der Importwert in ausländischer Währung ist. Da die Gültigkeit der Marshall-Lerner-Bedingung[5] vorausgesetzt wird, führt ein Devisenüberschuss, resultierend aus einer aktiven Handelsbilanz, zu einem Sinken des Wechselkurses (Aufwertung) am Devisenmarkt, bis ein neues Devisenmarktgleichgewicht bzw. ein Ausgleich der Handelsbilanz erreicht wird. Spiegelbildlich dazu verhält es sich bei einem Handelsbilanzdefizit, welches eine Abwertung der inländischen Währung zur Folge hat. Kritisch angemerkt werden muss jedoch, dass dieser Ansatz nur auf Stromgrößen abstellt und

[5] Zur Herleitung der Marshall-Lerner-Bedingung vgl. Borchert (2003).

Bestände nicht mit in die Analyse einbezieht. Des Weiteren werden der Kapitalverkehr und die Einkommensabhängigkeit der Importe außer Acht gelassen.
Letzterer Kritikpunkt wird in dem Einkommen-Ausgaben-Modell aufgegriffen. Dieser Ansatz richtet sein Hauptaugenmerk auf die Realeinkommensentwicklung und auf die aggregierte Nachfrage nach Gütern und Dienstleistungen als Determinanten des Wechselkurses. Es werden Störungen auf der Nachfrageseite – ausgelöst durch Realeinkommensänderungen in einer unterbeschäftigten Volkswirtschaft – und ihre Auswirkungen auf den Wechselkurs untersucht. Soll dieser Ansatz[6] weiter vervollständigt werden, so müssen die Gleichgewichtsbeziehungen für den Gütermarkt (11), den Geldmarkt (12) sowie die Bestimmungsgleichung für den Devisenbilanzsaldo (13) mit einbezogen werden (vgl. Jarchow (2000), S.160):

$$Y = C(Y) + I(i) + G + A(w, Y, Y_a) \tag{11}$$

$$mB = L(i, Y) \tag{12}$$

$$Z = A(w, Y, Y_a) + K(i, i_a), \tag{13}$$

wobei $C(Y)$ einkommensabhängiger Konsum, $I(i)$ zinsabhängige Investitionen und G die Staatsausgaben sind. Exporte und Importe werden zum Außenbeitrag A und Kapitalimporte und –exporte zu Nettokapitalimporte K saldiert und mit m wird der Geldangebotsmultiplikator und mit B die monetäre Basis (Geldbasis) bezeichnet. Durch Hinzunahme des monetären Sektors werden Zinsänderungen berücksichtigt und es kann somit auch zu einem zinsinduzierten Kapitalverkehr kommen. Bei flexiblen Wechselkursen ohne Devisenmarktinterventionen seitens der Zentralbank muss die Summe aus dem Außenbeitrag und dem Nettokapitalimport gleich Null sein.
Währungsreserven im Zusammenhang mit Interventionen der Währungsbehörde auf dem Devisenmarkt sind zur Stabilisierung des Wechselkurses von Interesse. Sobald die Zentralbank am Devisenmarkt interveniert, d.h. ausländische Währung an- bzw. verkauft, erhöht bzw. vermindert sich ihr Bestand an Währungsreserven. Aus diesem Zusammenhang ergibt sich, dass eine Verpflichtung der Zentralbank zur Verteidigung fester Wechselkurse die Kontrolle über die Geldbasis und damit das Geldangebot gefährdet. Um den Wechselkurs stabil zu halten, muss die Zentralbank ggf. bereit sein alle Devisen, die ihr angeboten werden, zu einem festegelegten Kurs anzukaufen, d.h. in entsprechendem Umfang heimische Währungseinheiten abzugeben. Falls beim festgelegten Wechselkurs ein Überschussangebot an Devisen besteht, nimmt die Zentralbank dieses Überschussangebot aus dem Markt und erhöht damit zunächst einmal die Geldbasis und damit das Geldangebot. Durch den Einsatz ihres Instrumentariums kann die Zentralbank versuchen, das durch den Devisenankauf geschaffene Geld wieder abzuschöpfen. Sie versucht dann, den Einfluss von Devisenbewegungen auf die Geldbasis und das Geldangebot zu neutralisieren (Neutralisierung- oder Stabilisierungspolitik).

2.5 Postkeynesianische Wechselkurstheorie

Im Gegensatz zu den bisher beschriebenen Ansätzen wird in dem Grundmodell der Portfoliotheorie zur Bestimmung des Wechselkurses bei den Finanzaktiva nicht mehr von perfekten, sondern von imperfekten Substituten und damit einhergehend auch nur noch von einer imperfekten Kapitalmobilität ausgegangen. Begründet wird dies mit der Annahme, dass die Anleger risikoavers sind und die ausländischen Finanzanlagen mit einem höheren Risiko behaftet einschätzen als inländische. Gemäß der optimalen Portfoliotheorie werden die ihr Vermögen international diversifizierenden Wirtschaftssubjekte diejenige Struktur ihres Vermögens wählen, die unter den Gesichtspunkten Ertrag und Risiko den individuellen Präferenzen am ehesten entspricht (vgl. Jarchow (2003)). Hierbei muss jedoch unter der Berücksichtigung von Risikoüberlegungen eine Risikoprämie mit in das Investitionskalkül einbezogen werden. Da-

[6] Keynesianisches Grundmodell.

bei wird zwischen einer kurzen und langen Frist unter der Annahme des Klein-Länder-Falls differenziert. Für diesen Fall sind das ausländische Zinsniveau, das ausländische Preisniveau und das ausländische Einkommen exogene Größen. Zudem soll das inländische Geldangebot wieder nur von Inländern gehalten werden.
Ähnlich wie bei den monetären Modellen der Wechselkursbestimmung wird der Wechselkurs auf kurze Frist von dem Angebot und der Nachfrage auf den Märkten für Finanzaktiva bestimmt (vgl. MacDonald und Tayler (1991), S.7), wobei neben dem Geldmarkt noch explizit der Markt für Wertpapiere mit betrachtet wird. Dabei soll sich die Reaktion der Finanzmärkte auf exogene Störungen sehr schnell vollziehen, so dass sich die betrachteten einzelnen Märkte stets im Gleichgewicht befinden. Des Weiteren werden Vermögensänderungen - z.B. ausgelöst durch Leistungsbilanzsalden, Geldmengenänderungen und Änderungen des Bondbestandes - und daraus resultierende Vermögenseffekte explizit mit in das Analysemodell einbezogen.
Das gesamte private Finanzvermögen W setzt sich zusammen aus der inländischen Geldmenge M, dem inländischen Bondbestand B und dem Bestand ausländischer Wertpapiere in der Hand von Inländern ausgedrückt in inländischer Währung wF:

$$W = M + B + wF \tag{14}$$

Die Geldmenge M wird beeinflusst von Wertpapierkäufen oder -verkäufen der Zentralbank in Form vom Staatsschuldtiteln B am offenen Markt. Kann zudem der Staat seine Budgetdefizite durch eine Geldschöpfung finanzieren, so besteht hierdurch eine weitere Möglichkeit, die Geldmenge zu beeinflussen. Der Bondbestand B setzt sich annahmegemäß nur aus den Staatsschulden der Regierung zusammen, die sich in Form von Wertpapieren in der Hand von inländischen Wirtschaftssubjekten befinden. Der Bestand ausländischer Wertpapiere in der Hand von Inländern F entspricht den kumulierten Leistungsbilanzüberschüssen der Vergangenheit gegenüber dem Ausland. Der ausländische Zinssatz i_a ist entsprechend dem Fall des kleinen Landes exogen. Die Marktgleichgewichtsbedingungen für die drei Finanzaktiva lauten wie folgt:

$$M = l\left(\bar{i}, \bar{i}_a + \beta\right)W , \tag{15}$$

$$B = b\left(\overset{+}{i}, \bar{i}_a + \beta\right)W , \tag{16}$$

$$wF = f\left(\bar{i}, \overset{+}{i}_a + \beta\right)W , \tag{17}$$

wobei $l + b + f = 1$ gilt und β die erwartete Wechselkursänderungsrate ist.
Hierbei sind auf der linken Seite die Bestände und auf der rechten Seite die Bestandsnachfragefunktionen der jeweiligen Finanzaktiva aufgeführt. Die Nachfrage nach den entsprechenden Finanzaktiva hängt dabei zum einen positiv und proportional von der Höhe des Vermögens W ab. Zum anderen wird die Nachfrage durch die Ertragsraten in- und ausländischer Wertpapiere i und i_a determiniert.
Die Gleichung (15) besagt, dass im Geldmarktgleichgewicht das Geldangebot der Geldnachfrage entspricht, wobei sich letztere aus einem bestimmten Anteil l des Gesamtvermögens W ergibt, den inländische Wirtschaftssubjekte an Kasse zu halten wünschen. Die Gleichung (16) bringt zum Ausdruck, dass im Gleichgewicht auf dem inländischen Bondmarkt das Bestandsangebot der Bestandsnachfrage entsprechen muss. Letztere wiederum macht einen bestimmten Anteil b des Gesamtvermögens W aus, den inländische Wirtschaftssubjekte in Form von inländischen Wertpapieren halten möchten. Vermögensänderungen ausgelöst durch Kursänderungen sollen durch die Annahme ausgeschaltet werden, dass auf dem Bondmarkt nur Wertpapiere mit fixen Kursen, aber einem variabel gestaltetem Nominalzins gehandelt werden. Gleichung (17) beschreibt das Gleichgewicht für den Markt von ausländischen Wertpapieren. Demnach muss im Gleichgewicht das Bestandsangebot der Bestandsnachfrage entsprechen, wobei letztere wieder dem Anteil f ausländischer Wertpapiere an dem Gesamtvermögen entspricht, den die inländischen Wirtschaftssubjekte zu halten bereit sind.
Über den Ausdrücken in den Klammern steht das Vorzeichen der jeweiligen partiellen Ableitung der Funktion nach der entsprechenden Variablen. Steigt z.B. der Inlandszins, so wird die inländische Geld- und die ausländische Bondnachfrage aufgrund der einsetzenden Substitution von Geld und ausländischen Wertpapieren gegen inländische Wertpapiere sinken, da eine Bondanlage im Inland attraktiver geworden ist. Somit stehen die inländische Bondnachfrage (inländische Geldnachfrage, ausländische Bondnachfrage) und der Zins in einem positiven (negativen) Verhältnis. Steigt dagegen der Auslandszins, so gehen infolge des einsetzenden Substitutionsprozesses inländische Geld- und Bondnachfrage zurück, da eine Auslandsanlage relativ gesehen günstiger ist. Die Koeffizienten l, b und f in den obigen drei Gleichungen geben die Anteile der Bestandsgrößen der einzelnen Aktiva am Gesamtvermögen an. Ihre Höhe hängt vom in- und ausländischen Zins (i und i_a) und der erwarteten Wechselkursänderung β ab.
Werden kurzfristige Reaktionen der Finanzmärkte auf exogene Störungen und die Auswirkungen auf die Wechselkursentwicklung untersucht, so hat z.B. eine expansive Geldpolitik eine Abwertung der inländischen Währung zur Folge. Hierdurch werden jedoch mittelfristig weitere Anpassungsreaktionen im realen Bereich der Volkswirtschaft ausgelöst, die sich langfristig wiederum auf die Finanzmärkte auswirken. Dieser dynamische Anpassungsprozess an das langfristige Gleichgewicht, in dem der Wechselkurs das Niveau erreicht hat, welches die Leistungsbilanz ins Gleichgewicht bringt soll hier nicht weiter verfolgt werden, kann aber unter Jarchow (2000), S.338ff nachgelesen werden.

3 Neuronale Netze

Im Idealfall lernen neuronale Netze ähnlich wie ein Gehirn an Beispielen. In künstlichen neuronalen Netzen werden einige Strukturen eines Nervensystems in karikativer Weise imitiert, um so ein Programm zu erhalten, mit dem Daten in einer bestimmten Weise verarbeitet werden können. Ein künstliches neuronales Netz besteht aus einer Menge von Knoten und deren Verbindungen untereinander, wobei jeder Knoten eine einzelne Nervenzelle modelliert. Vereinfacht ist ein neuronales Netz ein gerichteter und gewichteter Graph. Jeder Knoten j wird durch eine Variable $a_j(t)$ zum Zeitpunkt t beschrieben, die seinen Aktivierungszustand anzeigt. Für jede Verbindung zwischen zwei Knoten wird eine weitere Variable w_{ij} eingeführt, die die Stärke der Verbindung zwischen den Nervenzellen modelliert und als das Gewicht von Neuron i nach Neuron j bezeichnet wird (vgl. Abbildung 5).
Aus ökonomischer Sicht kann die Funktionsweise eines künstlichen Neurons auch als ein Entscheidungsmodell (Finnoff, Hergert und Zimmermann (1993)) interpretiert werden. So sammelt beispielsweise ein Devisenhändler vor einer Entscheidung über einen Kauf oder Nicht-Kauf einer bestimmten Währung zahlreiche Informationen unterschiedlicher Herkunft, die beispielsweise aus dem Bereich der technischen Analyse oder der Volkswirtschaft kommen können. Diese Informationen wird er individuell gewichten und aufsummieren, um so zu einem Gesamteindruck zu gelangen. Reicht dieser aus, einen bestimmten Schwellenwert zu übertreffen, so wird der Devisenhändler zu einer Kaufentscheidung gelangen. Im umgekehrten Fall wird er nicht reagieren. Hierbei wird deutlich, dass es sich bei dem beschriebenen Entscheidungsprozess letztendlich um ein schalterartiges, nicht-lineares Entscheidungsproblem handelt, welches durch eine nichtlineare Funktion in einem neuronalen Netz modelliert werden kann.

3.1 Neurosimulator FAUN

Die Entwicklung des Neurosimulators FAUN begann 1997 an der TU Clausthal und wird mit der FAUN-Projektgruppe an der Universität Hannover weitergeführt[7]. Heute ist es mit FAUN Release 1.0 komfortabel möglich, Probleme des überwachten Lernens mit künstlichen neuronalen Netzen zu lösen. Als Netze sind so genannte 3- und 4-lagige Perzeptrons und Radial-Basis-Netze mit und ohne Direktverbindungen verfügbar (vgl. Abbildung 1). Direktverbindungen zwischen der Eingabeschicht und der Ausgabeschicht erhöhen die Flexibilität eines künstlichen neuronalen Netzes. Es können „schwach nichtlineare“ Abhängigkeiten in den Ein- und Ausgabezusammenhängen leichter und besser approximiert werden. Im Vergleich zu anderen Neurosimulatoren trainiert FAUN Netze extrem schnell und konvergiert, dank globaler Optimierung, sehr zuverlässig (Breitner (2003)). Für FAUN 1.0 ist eine sehr komfortable, graphische Benutzeroberfläche unter Microsoft Windows und LINUX verfügbar. Mit der Benutzeroberfläche kann das Training der künstlichen neuronalen Netze einfach gesteuert und überwacht werden. Ferner können die besten trainierten Netze einfach ausgewählt und durch Bereitstellung des C- und FORTRAN-Quellcodes evaluiert werden.

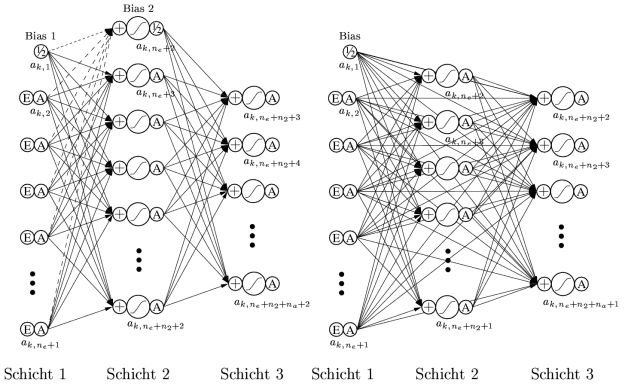


Abb. 1. Vollständig verbundenes dreilagiges Perzeptron ohne (links) bzw. mit Direktverbindungen (rechts) mit n_2 inneren Neuronen für eine n_e-dimensionale Eingabe $\boldsymbol{x}_k$ und eine n_a-dimensionale Ausgabe $f_{app}(\boldsymbol{x}_k; \boldsymbol{p})$.

3.2 Überwachtes Lernen

Überwachtes Lernen bedeutet, dass Ein-/Ausgabezusammenhänge $(\boldsymbol{x}_i, \boldsymbol{y}_i)$ - so genannte Muster - mit Input $\boldsymbol{x}_i \in \mathbb{R}^{n_e}$ und Soll-Output $\boldsymbol{y}_i \in \mathbb{R}^{n_a}$, $i = 1, 2, \ldots, n_m$, aus einem Musterdatensatz $\mathcal{D}_m$ gegeben sind, für die eine „möglichst gute“ $\mathcal{C}^r$-Approximationsfunktion $f_{app}(\boldsymbol{x}; \boldsymbol{p}^*)$ berechnet werden soll, wobei $f_{app}(\boldsymbol{x}; \boldsymbol{p}^*) : \mathbb{R}^{n_e} \times \mathbb{R}^{n_p} \to \mathbb{R}^{n_a}$ ist. $f_{app}(\boldsymbol{x}; \boldsymbol{p})$ hängt unendlich oft differenzierbar von $\boldsymbol{x}$ und dem wählbaren Parametervektor $\boldsymbol{p}$ ab. Dies ist u. a. wichtig für die Verwendbarkeit in der Praxis bzw. das Lösen schwieriger, multivariater Approximationsprobleme, wie z. B. Prognosen für Aktien, Indizes oder Zinsen sowie Kapitalmarktanalysen und -bewertungen (auch für Derivate). Dabei müssen die Muster in $\mathcal{D}_m$ problemgerecht auf den Trainingsdatensatz $\mathcal{D}_t := \{(\boldsymbol{x}_1, \boldsymbol{y}_1), \ldots, (\boldsymbol{x}_{n_t}, \boldsymbol{y}_{n_t})\}$ und den Validierungsdatensatz $\mathcal{D}_v := \{(\boldsymbol{x}_{n_t+1}, \boldsymbol{y}_{n_t+1}), \ldots, (\boldsymbol{x}_{n_m}, \boldsymbol{y}_{n_m})\}$ aufgeteilt werden. Wichtig ist eine Equilibrierung und Skalierung $\boldsymbol{x}_i \in [-1,1]^{n_e}$ und $\boldsymbol{y}_i \in [-c,c]^{n_a}$ mit $c \in]0,1[^+$ für alle Muster. Für das Training der neuronalen Netze wird in der Regel der Trainings- und Validierungsfehler

$$\varepsilon_t(\boldsymbol{p}) := \sum_{i=1}^{n_t} \sum_{k=1}^{n_a} \left(f_{app_k}(\boldsymbol{x}_i; \boldsymbol{p}) - y_{i,k} \right)^{2q},$$
$$\varepsilon_v(\boldsymbol{p}) := \sum_{i=n_t+1}^{n_m} \sum_{k=1}^{n_a} \left(f_{app_k}(\boldsymbol{x}_i; \boldsymbol{p}) - y_{i,k} \right)^{2q} \tag{18}$$

benutzt, wobei $q \in \mathbb{N}$ gelten muss und oft $q = 1$ verwendet wird.
Eine gute Approximationsfunktion $f_{app}(\boldsymbol{x}; \boldsymbol{p}^*)$ weist einen kleinen Fehler $\varepsilon_t(\boldsymbol{p}^*)$ pro Muster auf, d. h. $f_{app}(\boldsymbol{x}; \boldsymbol{p}^*)$ synthetisiert die Ein-/Ausgabezusammenhänge aus $\mathcal{D}_t$ ausreichend genau. Darüber hinaus ist ein gutes globales Approximations- bzw. Extrapolationsverhalten von $f_{app}(\boldsymbol{x}; \boldsymbol{p}^*)$ erforderlich. Dafür ist notwendig, dass auch der Fehler $\varepsilon_v(\boldsymbol{p}^*)$ pro Muster klein ist. In der Praxis muss $f_{app}(\boldsymbol{x}; \boldsymbol{p}^*)$ noch weiteren Anforderungen genügen, wie z. B. eine kleine Maximal- oder Gesamtkrümmung aufweisen (Breitner 2003)

4 Wechselkursprognosen mit neuronalen Netzen

Die zuvor vorgestellten Wechselkursmodelle sollen hier nicht im Einzelnen analysiert werden[8], sondern es finden aus allen Modellen verschiedenen Einflussgrößen Berücksichtigung bei der Prognose des Wechselkurses mit neuronalen Netzen. Der Vorteil dieser Vorgehensweise ist, das einzelne, in den Modellen isoliert betrachtete Größen, wie z.B. der Zins, nicht nur den Wechselkurs, sondern auch übergreifend Größen der anderen Modelle beeinflussen. Da neuronale Netze in der Lage sind Muster in den Eingabegrößen zu erkennen, wird somit ein breiterer Ansatz zur Wechselkursbestimmung geliefert, als es die einzelnen Modelle alleine könnten[9].

4.1 Thai Baht-US-Dollar Wechselkurs

Anhand des Thai Baht-US-Dollar Wechselkurses soll verdeutlicht werden, wie Wechselkursprognosen mit neuronalen Netzen erstellt werden können. Dazu wird der Zeitraum von Januar 1977 bis Juli 2004 betrachtet (siehe Abbildung 2). Der Wechselkurs ist gerade am Anfang der Zeitreihe (Januar 1977 bis Oktober 1978) und insbesondere über den langen Zeitraum von Mai 1981 bis Oktober 1984 konstant. Dies lässt auf ein Intervenieren des Thailändischen Staates durch geldpolitische Maßnahmen schließen, um den Wechselkurs nach einem kurzfristigen Anstieg auf einem bestimmten niedrigeren Niveau zu halten. Nach Freigabe des Wechselkurses pegelt dieser sich durch Angebot und Nachfrage an Devisenmärkten[10] auf einem höheren Niveau ein (vgl. hierzu auch Abbildung 5 zur prozentualen Änderung). Auffällig sind der starke Anstieg des Wechselkurses ab Juli 1997 und die starken Schwankungen danach. Dies ist auf die so genannte Asienkrise zurückzuführen, die in Thailand ihren Ursprung hatte und schnell auf mehrere asiatische Staaten übergriff. Als Asienkrise wird die Finanz- und Wirtschaftskrise Ostasiens der Jahre 1997 und 1998 bezeichnet. Auslöser dieser Krise waren ausländische Investoren, die, angezogen durch die sichtbar positive wirtschaftliche

[7] Siehe auch www.iwi.uni-hannover.de/faun.html.

[8] In Grimm (1997) werden verschiedene traditionelle Wechselkursmodelle auf ihre eventuelle Fehlspezifikation hin mit neuronalen Netzen jeweils einzeln untersucht.

[9] Der Autor erhofft sich somit genauere Prognosemodelle zu erstellen im Vergleich zu Grimm (1997).

[10] Bleiben Interventionen der Zentralbank außer Betracht, entstehen Angebot und Nachfrage nach Devisen aus Zinsarbitragegeschäften, Spekulationsgeschäften und Außenhandelsgeschäften, es kann also eine vollkommen freie Wechselkursbildung unterstellt werden.

Entwicklung in der ASEAN[11] –Region, an ein dauerhaftes Wachstum glaubten. Insgesamt 350 Milliarden US-Dollar flossen bis Ende 1996 hauptsächlich als Kredite in regionale Banken. Als die Erwartungen nicht bestätigt wurden, gingen von Thailand regelrechte Panikverkäufe aus, die durch eine nicht angemessene Wirtschaftspolitik verstärkt wurde und somit die Aktien und Preise immer weiter vielen. Die Stabilisierenden Maßnahmen des Internationale Währungsfonds (IWF) hatten wenig Erfolg. Experten kritisieren das Einschreiten des IWF sogar, weil er die betroffenen Länder zwang, den Leitzins sehr stark anzuheben, um den Fall der Wechselkurse entgegen zu wirken. Dies soll aber der realen wirtschaftlichen Entwicklung sehr stark geschadet haben und die Krise wurde somit verstärkt.

Abb. 2. Verlauf des Thai Baht-US-Dollar Wechselkurses von Januar 1977 bis Juli 2004. Deutlich ist die Auswirkung der Asienkrise (Beginn: Juli 1997), in Form eines starken Wechselkussanstieges zu erkennen.

4.2 Prognose des Thai Baht-US-Dollar Wechselkurses

Wie zuvor erwähnt sollen bei der Wechselkursprognose mit neuronalen Netzen verschiedene Einflussgrößen der zuvor vorgestellten Modelle Berücksichtigung finden. Die Schwierigkeit, die dabei besteht ist, das einige der Daten als Zeitreihen nicht zur Verfügung stehen bzw. manche nur auf Monatsbasis bezogen werden können. Während der Wechselkurs auf Tagesbasis verfügbar ist, wird z.B. der Geldmarktzins Thailands und der Leitzins der Vereinigten Staaten nur auf Monatsbasis angegeben[12]. Zudem ist der Geldmarktzins Thailands erst ab Anfang 1977 verfügbar. Daher werden Modelle auf Monatsbasis mit zwei verschiedene Prognosehorizonten erstellt: ein und zwei Monate.

[11] Association of South-East Asian Nations

[12] vgl. IFS – IWF International Financial Statistics: Leitzins United States: 11160B..ZF...

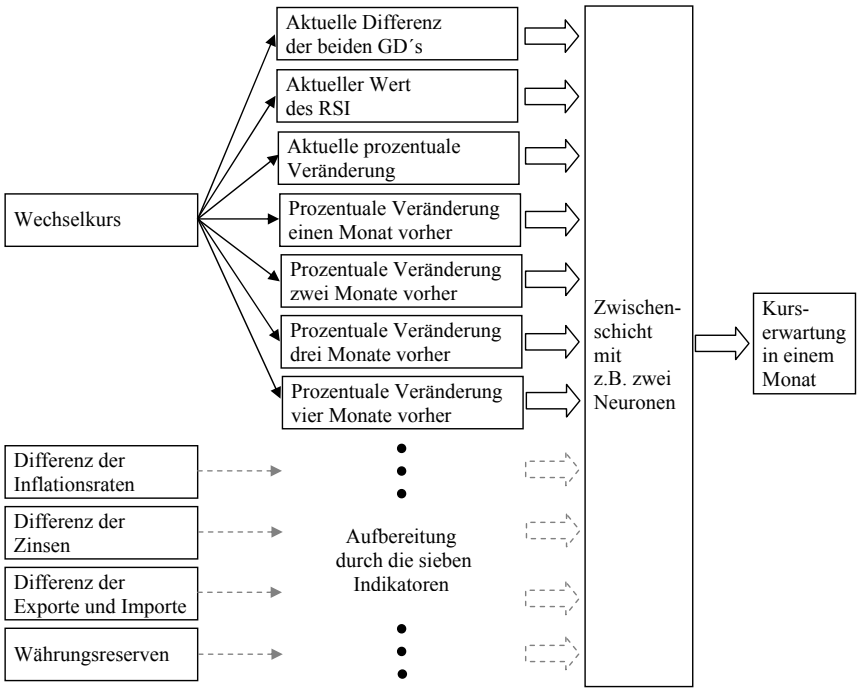


Abb. 3. Aufbereitung der fünf Inputwerte durch die sieben Indikatoren. Dadurch entsteht ein neuronales Netz, das 35 Eingabeneuronen hat und ein Ausgabeneuron.

4.2.1 Indikatoren und Aufbereitung der Muster

In das Prognosemodel geht nicht nur der Wechselkursverlauf, verstärkt durch verschiedene technische Indikatoren, ein, sondern auch die jeweiligen Inflationsraten, in- und ausländische Zinsen, Volumen von Exporten und Importen und die Währungsreserven Thailands.
Die Differenz der Inflationsraten wird betrachtet, da bei flexiblen Wechselkursen näherungsweise die prozentuale Änderungsrate des Wechselkurses mit der Differenz zwischen der Inflationsrate im Inland und der Inflationsrate im Ausland übereinstimmen muss, wenn die Kaufkraftparität in ihrer absoluten oder komparativen Form gilt[13]. Zusätzlich fließt die Differenz aus dem Geldmarktzins Thailands und dem Leitzins der Vereinigten Staaten, aber auch die Differenz der Exporte und Importe Thailands in das Modell ein.
Da neuronale Netze in der Lage sind komplexe Strukturen zu erkennen, werden alle Input-Werte der Zeitreihen mit folgenden technischen Indikatoren aufbereitet (vgl. Müller und Linder (2004) S.667ff):
Gleitende Durchschnitte (GD) dienen als Trendfolge-Indikatoren um brauchbare Handelssignale in Trendmärkten zu liefern. Hier wird die Differenz eines langen GD (von 12 Monaten), der dazu dient Marktbewegungen zu glätten, und einem kurzfristigen GD (von 6 Mona-

[13] Statistische Untersuchungen zeigen jedoch, dass Übereinstimmungen zwischen Wechselkursänderungsraten und Inflationsdifferenz gering sind, da die realen Wechselkurse erst langfristig gesehen zur Kaufkraftparität tendieren, vgl. Jarchow (2000) S.272ff.

ten), der mit den Kreuzungen des langfristigen „Moving Average“ Handelssignale generiert, benutzt.
Der **Relative Stärke Index (RSI)** liefert als Oszillator Handelssignale nur bei Seitwärtsbewegungen[14]. Hier wird eine kurzfristige Einstellung (von 6 Monaten) herangezogen[15].
Zu diesen beiden Inputvariablen kommen für jede Zeitreihe als zusätzliche Inputvariablen jeweils die **prozentualen Veränderungen** der letzten vier Monate, so dass jeder der fünf Input-Werte mit sieben Indikatoren versehen wird und damit das neuronale Netz aus 35 Eingabeneuronen besteht (siehe Abbildung 3).
Prognostiziert werden soll die prozentuale Wechselkursänderung in einem bzw. zwei Monaten. Das Ausgabeneuron enthält dann die Prognose, ob der Kurs des Basistitels im folgenden Monat (bzw. dem folgenden zweiten Monat) steigt oder fällt. Die Höhe dieses Wertes kann gleichzeitig auch als Stärke des Handelssignals interpretiert werden.

Tabelle 1. Detaillierte Auswertung des Trainings zur Prognose des Wechselkurses in einem Monat.

Topologie	**I**	**II**	**III**	**IV**	**V**	**VI**	**VII**	**VIII**	**IX**	**X**
innere Neuronen	1	1	2	2	3	3	4	4	5	5
Direktverbindungen	nein	ja	nein	ja	nein	ja	nein	ja	nein	ja
Anzahl der Gewichte	38	73	75	110	112	147	149	184	186	221
ε_t*	0.9598	0.6105	0.6586	0.6035	0.4735	0.4313	0.4009	0.3653	0.2811	0.3849
$n_t/n_v\ \varepsilon_v$*	0.0479	0.1124	0.0486	0.4201	0.0536	0.1435	0.0449	0.1362	0.0459	0.0959
durchs. Train.-Fehler	0.0941	0.0750	0.0779	0.0746	0.0661	0.0630	0.0608	0.0580	0.0509	0.0596
proz. Train.-Fehler	0.0495	0,0395	0.0410	0.0393	0.0347	0.0332	0.0319	**0.0305**	**0.0268**	**0.0313**
Anzahl zufällig initialisierter Netze	103	113	215	222	309	347	416	512	534	683
Anzahl erfolgreich trainierter Perzeptrons	100	100	200	200	300	300	400	400	500	500
% nicht erfolgreich trainierter Netze	2.91%	11.5%	6.98%	9.91%	2.91%	13.54%	3.85%	21.88%	6.37%	26.79%
Rechenzeit in sec.	26.5	78.5	1051	237.4	302.5	533.5	711.9	1203.4	1297.2	2247.5

4.2.2 Auswertung des Trainings der Topologien

Die so durch die Aufbereitung mit den technischen Indikatoren entstandenen 319 Muster werden für das Training und die anschließende Auswertung in einen Trainings-, Validierungs- und Generalisierungsdatensatz aufgeteilt, wobei die letzten 51 Muster am Ende der Zeitreihe für die Generalisierung benutzt werden. Bei dem Training der neuronalen Netze stellte sich schnell heraus, dass die Aufteilung der übrigen 268 Muster in Trainings- und Validierungsmuster nicht trivial ist[16]. Werden z.B. zu große Bereiche der Asienkrise oder Bereiche zu den Zeitpunkten fester Wechselkurse (vgl. Abbildung 2), bei der Validierung berücksichtigt und somit aus den Trainingsdaten heraus geschnitten, dann werden nur Netze mit einem sehr hohen Trainingsfehler gefunden. Erst eine Aufteilung in 217 Trainings- und 51 Validierungsmuster, wobei die Validierungsmuster aus drei zusammenhängenden Blöcken bestehen, liefert einen besseren Trainings- und Validierungsfehler (ε_t* und ε_v*) für alle Topologien (vgl. Tabelle 1). Es kann also erhofft werden, dass alle Topologien bei so geringen Fehlern den Wechselkurs hinreichend genau erlernen und prognostizieren. Die Anzahl der inneren Neuronen variiert von eins bis fünf. Ab sechs inneren Neuronen wäre die Anzahl der Gewichte im Vergleich zu den Trainingsdaten zu hoch und es liegt „kein gut gestelltes“ Funktionsapproximationsproblem vor (vgl. Breitner (2003)).
Bei der visuellen Betrachtung der prognostizierten Wechselkurse in einem Monat und in zwei Monaten[17] stellte sich jedoch heraus, dass Topologien mit Direktverbindungen versuchen, die Wechselkursschwankungen zu erlernen, während Topologien ohne Direktverbindungen immer eine waagerechte Gerade in den Daten ausbilden und nur auf sehr starke Ausschläge reagieren, vgl. Abbildung 5 und 6. Insbesondere ist zu erkennen, dass alle Topologien auf die Asienkrise reagieren und diese sehr starken Schwankungen, zusätzlich zu dem einzelnen Ausschlag nach der Periode fester Wechselkurse, sehr gut erlernen. Zudem weisen alle Topologien ein bis drei starke Ausschläge immer an den selben Stellen in dem Generalisierungsdatensatz auf, sodass sie, nachdem die Asienkrise erlernt wurde, in der Zukunft weitere Krisen an den gleichen Stellen prognostizieren, die aber nicht eintreten (vgl. Anhang A).
Um eventuell bessere Prognosen mit Direktverbindungen zu liefern, wird eine Expertenrundentopologie jeweils für die Prognose in ein und zwei Monaten erstellt, bestehend aus den fünf Topologien mit Direktverbindungen. Dabei gehen die einzelnen Topologien gewichtet und dann aufsummiert in das Modell ein. Jedes der fünf Gewichte wird mit

$$g_T = \frac{1-\varepsilon^*_{t,T}}{\sum_{k=1}^{5} 1-\varepsilon^*_{t,k}} \quad \text{mit } T = 1,\ldots,5 \tag{19}$$

berechnet. Wie in Abbildung 4 zu erkennen ist liefert dies für den Generalisierungsdatensatz bedingt bessere Ergebnisse für die zukünftige Prognose als nur mit einer einzelnen Topologie (vgl. dazu auch Abbildung 5 und die Abbildungen 10 bis 13 im Anhang).

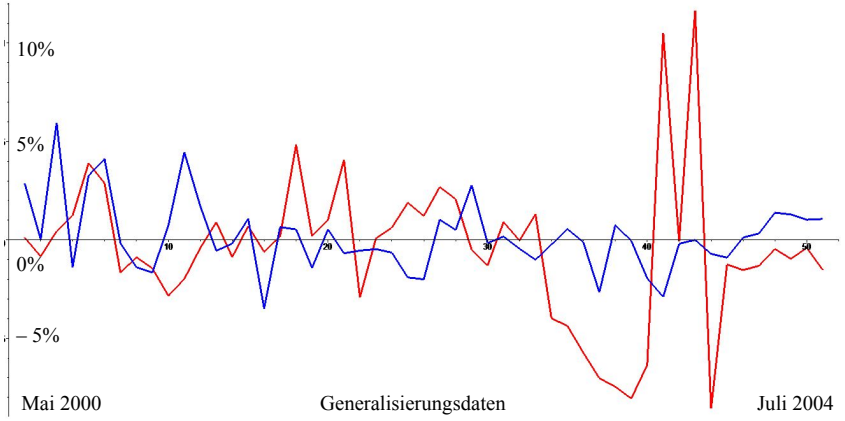


Abb. 4. Vergleich der prozentualen Wechselkursänderungsrate in einem Monat (blau) gegenüber der Prognose (rot) mit der Expertenrundentopologie. Betrachtet wird hier nur der Generalisierungsdatensatz. Auch hier wird eine Krise am Ende der Zeitreihe vorhergesagt (vgl. die Ausreißer am Ende).

[14] Zielsetzung ist hier, die Fehlsignale zu minimieren, indem vom neuronalen Netz vorgegeben werden soll, welcher der beiden Indikatoren (GD oder RSI) gerade zu beachten ist. Sofern der Basistitel von einem Trendmarkt zu einem Seitwärtsmarkt wechselt, sind also nur die Signale des Oszillators zu beachten. Analog beim Trendfolger, vgl. Müller und Linder (2004) S.676f.
[15] Zur genauen Berechnung und Herleitung des RSI siehe Müller und Linder (2004) S.343f.
[16] Dabei sollten Trainings- und Validierungsdaten so gewählt werden, dass $1 \le n_t/n_v \le 9$ gilt (vgl. Breitner (2003)).
[17] Vgl. auch die Abbildungen im Anhang A.

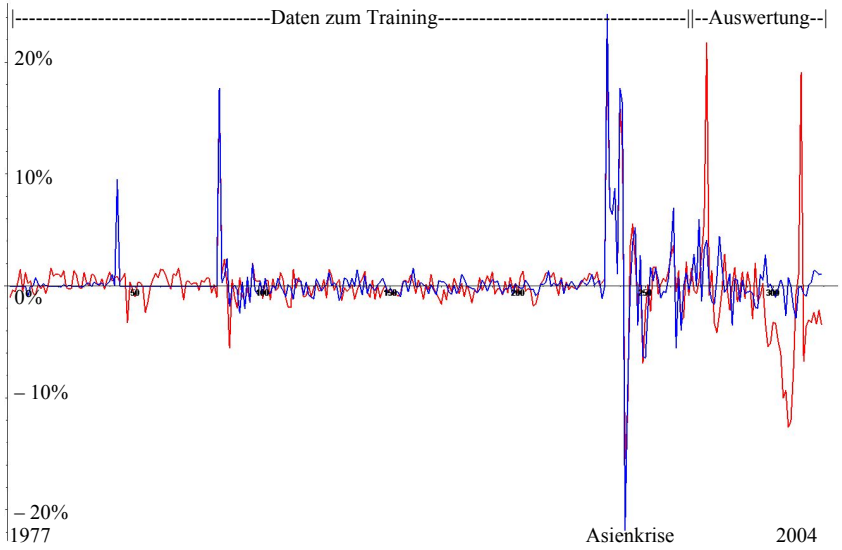


Abb. 5. Vergleich der prozentualen Wechselkursänderungsrate in einem Monat (blau) gegenüber der Prognose (rot) mit einem neuronalen Netz mit vier inneren Neuronen und Direktverbindungen. Betrachtet werden alle 319 Muster, also auch der Trainingsdatenraum und die Daten zur Auswertung.

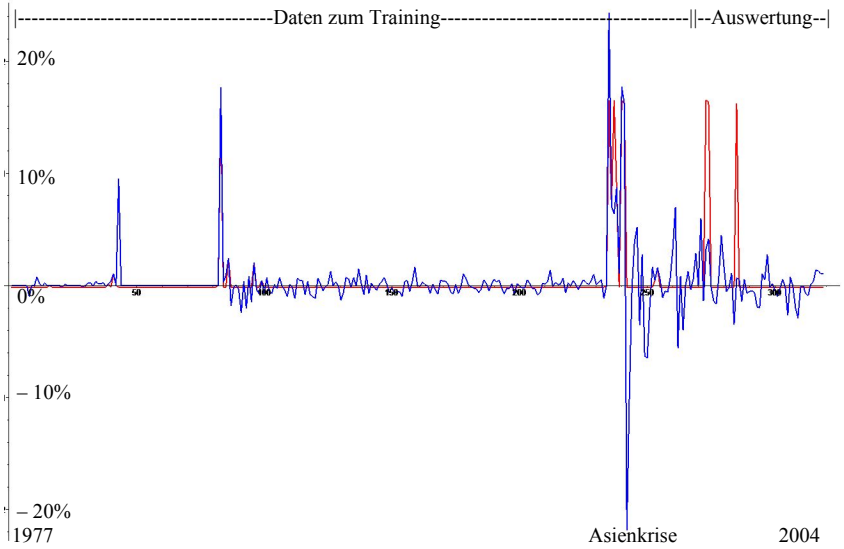


Abb. 6. Vergleich der prozentualen Wechselkursänderungsrate in einem Monat (blau) gegenüber der Prognose (rot) mit einem neuronalen Netz mit einem inneren Neuron ohne Direktverbindungen. Betrachtet werden auch hier alle 319 Muster. Das neuronale Netz bildet immer, mit Ausnahme der wenigen Schwankungen (vgl. z.B. die Asienkrise), eine Gerade zwischen den Daten aus.

4.2.3 Qualität der Prognosen

Die graphische Analyse ist sehr hilfreich, um zu erkennen, wie die neuronalen Netze in den Daten liegen und welche Topologien geeigneter bei der Prognose von Wechselkursen sind. Dennoch liefern diese Betrachtung und der gute Trainingsfehler der Netze noch keine Aussage darüber, wie gut nun diese Prognosen sind. Dazu werden die im Folgenden beschriebenen statistischen Gütekriterien (vgl. Grimm (1997) S.62ff) für jede Topologie ausgewertet (vgl. Tabelle 2 und 3).
Als erstes Gütemaß zur Bewertung des Prognosefehlers dient die Quadratwurzel aus dem mittleren quadratischen Fehler (Root Mean Square Error = RMSE):

$$\mathrm{RMSE} = \sqrt{\frac{1}{n} \cdot \sum \left(\mathrm{target}_t - \mathrm{out}_t\right)^2} \tag{20}$$

mit n = Gesamtzahl der Prognosen, target_t = Zielgröße des t-ten Musters und out_t = Prognosegröße des t-ten Musters.
Gegenüber dem mittleren quadratischen Fehler hat dieser den Vorteil, dass er die gleiche Dimension wie die Zielgröße und die Prognosegröße hat.
Als ein weiteres Gütekriterium dient der lineare Korrelationskoeffizient (r) nach Bravais-Pearson[18] zwischen der Ziel- und Prognosegröße, der sich aus der folgenden Gleichung ergibt:

$$\mathrm{r} = \frac{\sum_{t=1}^{n}\left(\mathrm{out}_t - \overline{\mathrm{out}}\right)\left(\mathrm{target}_t - \overline{\mathrm{target}}\right)}{\sqrt{\sum_{t=1}^{n}\left(\mathrm{out}_t - \overline{\mathrm{out}}\right)^2 \sum_{t=1}^{n}\left(\mathrm{target}_t - \overline{\mathrm{target}}\right)^2}} \tag{21}$$

Dieser misst die lineare Korrelation bzw. die Stärke des Zusammenhangs zwischen der prognostizierten und der tatsächlichen Größe gemittelt über alle Beobachtungen und kann Werte von minus bis plus Eins annehmen. Ein Wert von Eins würde eine perfekte Übereinstimmung der Bewegung zwischen der prognostizierten und der tatsächlichen Größe bedeuten. Bei einem Wert von Null existiert kein Zusammenhang und bei einem Wert von minus Eins verhält sich die Prognose genau entgegengesetzt zur Zielgröße.
Die Gleichung (22) der Wegstrecke (WS) berechnet, wie viel des bei perfekter Voraussicht maximal möglichen Gewinns pro Kapitaleinheit durch die Prognose theoretisch erzielt wird. Handelt es sich bei dem Modell um eine Prognose des Niveaus der Zielgröße, so wird im Zähler die tatsächliche Bewegung der Zielgröße mit dem Vorzeichen der prognostizierten Bewegung multipliziert und anschließend aufsummiert. Im Nenner steht der maximal mögliche Gewinn pro eingesetzter Kapitaleinheit bei perfekter Voraussicht, indem die Absolutwerte der Bewegung der Zielgröße aufsummiert werden. Die Werte der Wegstrecke liegen in einem Intervall von [1,-1]. Bei einem Wert von Eins erzielt das Prognosemodell den maximal möglichen Gewinn. Spiegelbildlich verhält es sich bei einem Wert von minus Eins. Liegen die Werte um Null, so würde dies auf ein Random-Walk hindeuten. Für die Auswertung eines Modells, welches die prozentuale Veränderung der Zielgröße prognostiziert, ergibt sich die folgende Wegstreckenberechnung:

[18] Es lässt sich zeigen, dass der Absolutbetrag des Korrelationskoeffizienten der Quadratwurzel aus dem Bestimmtheitsmaß R^2 entspricht, so dass R^2 hier nicht weiter betrachtet wird.

$$\mathrm{WS} = \frac{\sum_{t=1}^{n-1} \left(\Delta\mathrm{target}_{t+1}\right) \cdot \mathrm{Vorzeichen}\left(\Delta\mathrm{out}_{t+1}\right)}{\sum_{t=1}^{n-1} \left|\Delta\mathrm{target}_{t+1}\right|} \tag{22}$$

Ist nur die zukünftige Richtung der Zielgröße entscheidend, so bietet sich die Berechnung der Trefferquote (TQ) bei einer Prognose der prozentualen Veränderung an:

$$\mathrm{TQ} = \frac{1}{n}\sum_{t=1}^{n-1} h_t \quad \text{mit} \quad h_t = \begin{cases} 1 \text{ wenn } \left(\Delta\mathrm{out}_{t+1}\right)\cdot\left(\Delta\mathrm{target}_{t+1}\right) > 0 \\ 0 \text{ wenn } \left(\Delta\mathrm{out}_{t+1}\right)\cdot\left(\Delta\mathrm{target}_{t+1}\right) < 0 \end{cases} \tag{23}$$

Für Werte von Eins bedeutet dies, dass das Modell alle Bewegungen der Zielgröße korrekt prognostiziert. Bei einem Wert von Null liegt das Modell in allen Fällen falsch. Ein Wert von 0.5 würde bedeuten, dass ein Münzwurf ein ebenso gutes Ergebnis erzielt hätte.
Der Ungleichheitskoeffizient (U) von Theil dient dazu, die Prognose des Modells mit der naiven Prognose vergleichen zu können. Für das Theilsche U ergibt sich bei einer Prognose der prozentualen Veränderung:

$$\mathrm{U} = \sqrt{\frac{\sum_{t=1}^{n} \left(\Delta\mathrm{target}_t - \Delta\mathrm{out}_t\right)^2}{\sum_{t=1}^{n} \left(\Delta\mathrm{target}_t - \Delta\mathrm{target}_{t-1}\right)^2}} \tag{24}$$

Die naive Prognose des Niveaus der Zielgröße geht davon aus, dass der aktuelle Kurs der bestmögliche Predikator für den zukünftigen Kurs ist. Für die Prognose der prozentualen Veränderung gilt analog, dass die letzte Veränderung des Kurses der zukünftig erwarteten Veränderung entspricht. Fällt dieser Fehler geringer aus als der des Prognosemodells, so würde U Werte von größer als Eins annehmen. Dies bedeutet, dass das Modell schlechtere Prognosen liefert als das naive. Bei Werten von U kleiner Eins schneidet das Prognosemodell dagegen besser ab.
Um zu einer Aussage über die Rentabilität des Modells zu gelangen, kann die monatliche Rendite ROI (Return on Investment) aus der gemittelten Summe der Einzelrenditen gemäß den folgenden Gleichungen bei einer Prognose der prozentualen Veränderung berechnet werden:

$$\mathrm{ROI} = \frac{1}{n-1} \cdot \sum_{t=1}^{n-1} \mathrm{Vorzeichen}\left(\Delta\mathrm{out}_{t+1}\right) \cdot \left(\frac{\mathrm{target}_{t+1} - \mathrm{target}_t}{\mathrm{target}_t}\right) \tag{25}$$

Die Gesamtauswertung der Modelle geschieht in der vorliegenden Arbeit in der folgenden Art und Weise. Das beste Modell wird anhand der statistischen Gütekriterien auf der Generalisierungsmenge mit einem Rangsummenverfahren separat ermittelt. Dabei wird demjenigen Modell ein höherer Rang zugewiesen, welches bei dem jeweiligen Gütekriterium auf der Generalisierungsmenge am besten abgeschnitten hat. Anschließend werden die einzelnen Ränge addiert und als jeweils bestes Modell jenes mit der geringsten Rangsumme ermittelt.
Werden die Topologien für die Prognose des Wechselkurses in einem Monat mit diesem Rangsummenverfahren betrachtet, so sind die beiden besten Topologien Netze ohne Direktverbindungen (vgl. Tabelle 2 1no und 4no) und erst auf dem dritten Platz liegt die Expertenrundentopologie. Das heißt, dass die Expertenrundentopologie insgesamt gesehen besser abschneidet als alle anderen Topologien mit Direktverbindung, aus denen sie generiert wurde. Alle Topologien weisen einen kleinen RMSE auf, sie weichen also nicht allzu stark im Mittel gesehen von dem tatsächlichen Wechselkurs ab. Bei dem Korrelationskoeffizienten (r) sieht das anders aus, alle Werte liegen nah bei Null, einige leicht drüber und einige darunter, so dass hier kaum eine Korrelation zu erkennen ist und somit fast keine Übereinstimmungen

herrschen. Das Gütekriterium der Wegstrecke (WS) gibt an, wie viel von dem maximalen Gewinn (Wert = 1) mit dem Prognosemodell möglich ist. Da alle Werte bis auf einen leicht über Null liegen wäre bei allen Modellen ein leichter Gewinn möglich, aber Werte um Null bedeuten, dass sich die Prognosen wie ein Random-Walk verhalten. Jedoch liegen bei einigen Modellen die Trefferquoten (TQ) höher als 50%, so dass dies einem Random-Walk widerspricht. In der Praxis gelten Prognoseverfahren die eine Trefferquote von mehr als 55% haben, als sehr gut und werden dementsprechend eingesetzt. An dieser Stelle sei aber schon darauf hingewiesen, dass hier nur 51 Muster für die Generalisierung benutzt werden, sodass diese Auswertung der Topologien kritisch zu würdigen ist. Es müssten mehr Daten für die Auswertung zur Verfügung stehen, die aber nicht von den Trainingsdaten abgezogen werden können, da dann nur noch schlechtere Netze trainiert werden. Alle Prognosen schneiden schlechter ab als die naive Prognose es täte (vgl. Tabelle 2), da alle Werte über Eins liegen. Bei der Rentabilität der Modelle (ROI) haben alle negative Renditen, so dass keines der Modelle als gut angesehen werden kann bzw. „sie zu oft falsch liegen“.

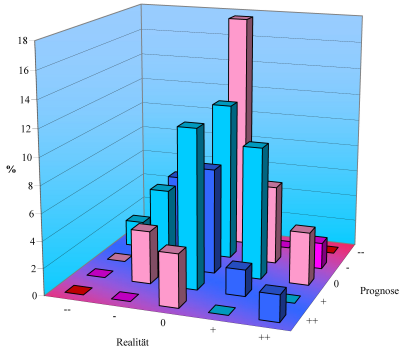
Tabelle 2. Statistische Gütekriterien für alle berechneten Topologien und der Expertenrundentopologie (Experten) bei der Prognose des Wechselkurses in einem Monat. 1no = 1 inneres Neuron ohne Direktverbindung, 1nm = 1 inneres Neuron mit Direktverbindung, usw.

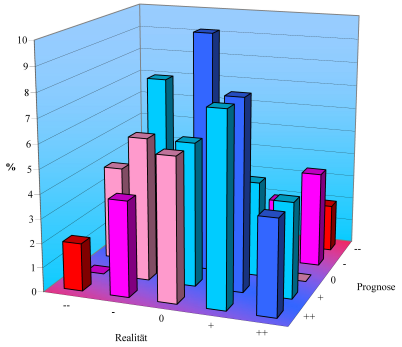
	RMSE		r		WS		TQ		U		ROI		RS	RANG
1no	0.0373	1	0.3503	1	0.1069	5	58.82%	2	1.5784	1	-23.5063	11	21	**1**
1nm	0.0476	3	0.0829	4	0.0746	7	50.98%	8	2.0189	3	-16.2316	7	32	**5**
2no	0.0867	10	0.0415	9	0.1388	4	52.94%	5	3.6899	10	-14.2916	2	40	8
2nm	0.0636	6	-0.0452	8	0.0193	10	50.98%	8	2.7093	6	-16.6596	8	46	10
3no	0.0764	8	-0.0539	7	0.2179	3	56.86%	3	3.2508	8	-14.1057	1	30	**4**
3nm	0.0613	5	-0.0810	5	-0.0587	9	47.06%	10	2.6037	5	-15.8635	4	38	7
4no	0.0759	7	0.0999	3	0.2338	2	52.94%	5	3.2291	7	-14.4638	3	27	**2**
4nm	0.0578	4	0.2420	2	0.0166	11	47.06%	10	2.4570	4	-16.0464	5	36	6
5no	0.1359	11	-0.0282	10	0.2897	1	60.78%	1	5.7921	11	-16.1502	6	40	8
5nm	0.0844	9	-0.0017	11	0.0591	8	56.86%	3	3.5961	9	-16.6838	9	49	11
Experten	0.0411	2	0.0545	6	0.0981	6	52.94%	5	1.7422	2	-16.6945	10	29	**3**

Tabelle 3. Statistische Gütekriterien für alle berechneten Topologien und der Expertenrundentopologie (Experten) bei der Prognose des Wechselkurses in zwei Monaten, wobei die Topologie mit einem inneren Neuron und ohne Direktverbindungen (1no) nicht mit in das Rangsummenverfahren eingeht. 1nm = 1 inneres Neuron mit Direktverbindung, usw.

	RMSE		r		WS		TQ		U		ROI		RS	RANG
1no	0.0834	X	0.0387	X	-0.0856	X	49.02%	X	3.4992	X	1.1060	X	X	X
1nm	0.2103	6	0.2275	4	0.1696	3	54.90%	3	8.8375	6	1.0357	3	25	5
2nm	0.1236	5	0.2625	3	0.1944	2	56.86%	2	5.1747	5	0.9945	4	21	4
3nm	0.0805	2	0.1555	5	0.0373	4	47.06%	4	3.3779	2	1.3009	1	18	**2**
4nm	0.0789	1	0.2803	1	0.3351	1	60.78%	1	3.3119	1	0.7291	6	11	**1**
5nm	0.1203	4	0.0194	6	-0.1261	6	43.14%	5	5.0528	4	0.8099	5	30	6
Experten	0.0854	3	0.2685	2	-0.0609	5	43.14%	5	3.5822	3	1.1420	2	20	**3**

Anders sieht es jedoch bei den Prognosen[19] der Wechselkurse in zwei Monaten aus, dort haben alle berechnetten Modelle einen positiven ROI-Wert (vgl. Tabelle 3). Dies ist insofern bemerkenswert, da meist kurzfristige Prognosen besser abschneiden als längerfristige. Auch liegt die Trefferquote bei drei Prognosen bei 55% bzw. sogar darüber. Und auch bei der Betrachtung der r und WS Werte kann zu dem Schluss gekommen werden, dass diese drei Modelle besser sind als ein Münzwurf. Die Expertentopologie liefert hier nicht so gute Werte und liegt insgesamt entsprechend nur auf Rang drei. Festzuhalten ist, dass alle Modelle trotz eines geringen RMSE-Wertes schlechtere Prognosen liefern würden als die naive Prognose. Es scheint also nach diesen Gütekriterien die Prognose in zwei Monaten besser abzuschneiden als die Prognose für einen Monat im Voraus.





%	Real.					
Prog.	--	-	0	+	++	ges.
--	0,00	3,92	17,65	0,00	0,00	21,57
-	1,96	5,88	11,77	5,88	1,96	27,45
0	0,00	5,88	7,85	9,80	3,92	27,45
+	0,00	3,92	11,77	1,96	0,00	17,65
++	0,00	0,00	3,92	0,00	1,96	5,88
ges.	1,96	19,60	52,96	17,64	7,84	100,00

%	Real.					
Prog.	--	-	0	+	++	ges.
--	1,96	3,92	5,88	1,96	1,96	15,68
-	0,00	3,92	3,92	0,00	3,92	11,76
0	3,92	7,84	9,84	3,92	0,00	25,52
+	0,00	5,88	5,88	7,84	3,92	23,52
++	1,96	3,92	5,88	7,84	3,92	23,52
ges.	7,84	25,48	31,40	21,56	13,72	100,00

Abb. 7. Vergleich der beiden besten (nach dem Rangsummenverfahren) Netze mit der Performance Matrix. Links: die Topologie mit einem inneren Neuron mit Direktverbindungen für die Prognose in einem Monat, rechts: die Prognose in zwei Monaten mit der Expertenrundentopologie.

Einen genauen Einblick in das jeweilige Modell bietet jedoch die Abbildung 7 und die Abbildungen 14 und 15 im Anhang A. Hier wird eine so genannte Performance Matrix verwendet, die Realität und Prognose gegenüberstellt. Hierbei werden folgende Beziehungen verwendet:

- ++, stark steigend, falls der Kurs um mehr als 3% steigt
- +, steigend, falls der Kurs um 1% bis 3% steigt
- **0**, gleich bleibend, falls der Kurs im Rahmen von +/– 1% gleich bleibt
- –, fallend, falls der Kurs um 1% bis 3% fällt
- – –, stark fallend, falls der Kurs um mehr als 3% fällt.

Steigt der Kurs z.B. stark und prognostiziert das neuronale Netz ebenfalls einen starken Anstieg, wird die Prognose dem Feld (++/++) zugerechnet. Bleibt der Kurs jedoch gleich und das neuronale Netz prognostiziert einen starken Anstieg, wird der Wert dem Feld (**0**/++) zugerechnet. Werte die auf der Diagonalen von (– –/– –) nach (++/++) liegen, sind exzellente

[19] Bei der Prognose des Wechselkurses in zwei Monaten werden die Topologien ohne Direktverbindungen nicht mehr betrachtet, da sie nur wenige hohe Ausschläge haben (also Krisen hervorsehen) und sonst nur eine Gerade in den Daten ausbilden.

Prognosen, da Realität und Prognose übereinstimmen. Sie sind blau gekennzeichnet. Werte, die auf den Nebendiagonalen liegen sind Prognosen, die von der Realität nur wenig abweichen. Sie sind cyan eingezeichnet. Je weiter der Wert von der Diagonale entfernt ist, umso schlechter ist die Prognose. Solche Werte sind in verschiedenen Rottönen gekennzeichnet (vgl. Mettenheim (2003)). Für die besten Modelle der Prognosen für die Kursänderung in einem Monat und in zwei Monaten wird eine Performance Matrix berechnet und zur Veranschaulichung als Diagramm ausgegeben (vgl. Anhang A Abbildungen 14 und 15).
Die Prognosen des Wechselkurses in einem Monat liefert grundsätzlich bessere Ergebnisse als die Prognosen des Kurses in zwei Monaten, da kaum zu stark abweichende Prognosen erstellt werden, somit ist die Diagonale von (– –/– –) nach (++/++) und deren Nebendiagonalen besser besetzt (vgl. Abbildung 7). Hier liegen anteilig gesehen fast alle Werte, während bei den Prognosen des Kurses in zwei Monaten auch stark entgegengesetzte Prognosen erstellt werden.

4.6 Besondere Betrachtung der Asienkrise

Zum Abschluss soll die Frage geklärt werden, ob es möglich gewesen wäre die Asienkrise mit neuronalen Netzen hervorzusagen. Dazu werden 49 Muster, beginnend ab Juli 1997 (also ab Anfang der Asienkrise) bis einschließlich Juli 2001, aus dem gesamten Datensatz für eine anschließende Generalisierung herausgeschnitten. Die übrigen Muster werden zum Training benutzt. Dabei werden die gleichen Analysen des Trainings und anhand der Gütekriterien (vgl. Tabelle 5 und 6 im Anhang B) durchgeführt. Es ist festzustellen, dass, wenn die Asienkrise keine Berücksichtigung findet, ähnlich gute, wenn nicht sogar bessere Werte bei den Gütekriterien erzielt werden (vgl. die Werte für TQ und ROI in Tabelle 3 und 6). Auch werden bessere Prognosen, in Hinsicht auf die Performance Matrizen (vgl. Abbildung 16 im Anhang A), als bei den Prognosen für den Kursverlauf in einen Monat und in zwei Monaten, erstellt. Allerdings werden hier, da wir es mit stärkeren Schwankungen von bis zu +/– 20% des Wechselkurses zu tun haben, die Beziehungen für die Performance Matrix wie folgt angepasst:

- ++, stark steigend, falls der Kurs um mehr als 6% steigt
- +, steigend, falls der Kurs um 2% bis 6% steigt
- **0**, gleich bleibend, falls der Kurs im Rahmen von +/– 2% gleich bleibt
- –, fallend, falls der Kurs um 2% bis 6% fällt
- – –, stark fallend, falls der Kurs um mehr als 6% fällt.

Aus diesen bisherigen Ergebnissen lässt sich schließen, dass neuronale Netze, die ohne die Asienkrise trainiert werden, bessere Prognosen liefern, insbesondere auch für die Asienkrise und die Werte danach.
Allerdings liefert eine graphische Analyse der tatsächlichen und prognostizierten Wechselkursänderungsrate ein ernüchterndes Ergebnis: keine der Topologien sagt den sofortige starken Anstieg des Kurses voraus, sondern sie reagieren wenn überhaupt erst zeitverzögert (vgl. Abbildung 8 und 9 und die Abbildung 17 im Anhang), dann aber auch mit ähnlich starken Schwankungen. Das heißt, eine Krise wird zwar erkannt, aber erst, wenn sie schon da ist.

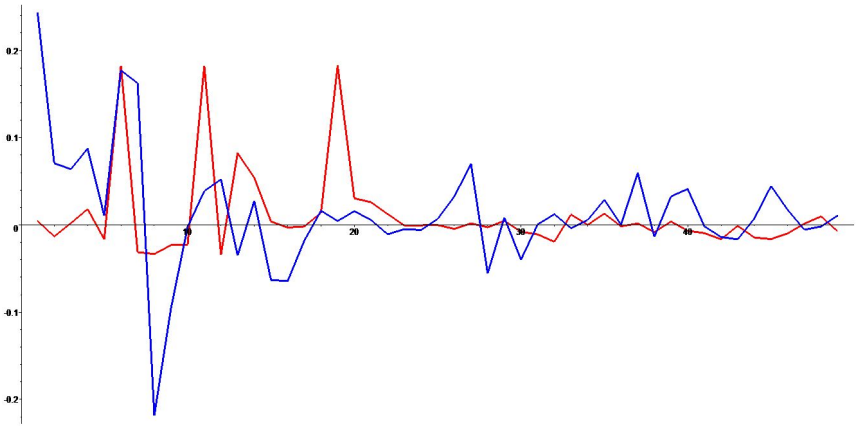


Abb. 8. Dargestellt sind hier die prozentualen Wechselkursänderungen in einem Monaten (blau) während der Asienkrise und der Zeit danach (49 Muster von Juli 1997 bis Juli 2001) gegenüber der Prognose (rot) mit einem neuronalen Netz mit einem inneren Neuronen. Eine Krise, also der sofortige starken Anstieg des Kurses, wird erst zeitverzögert wahrgenommen.

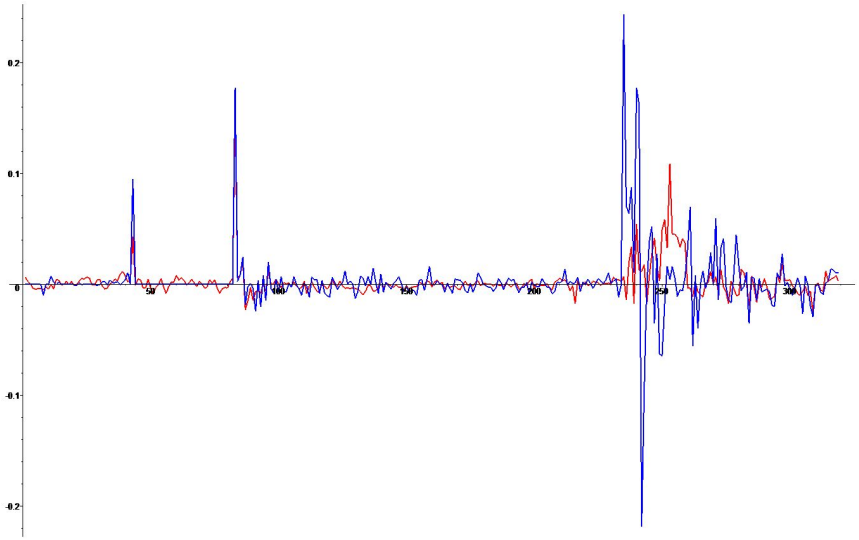


Abb. 9. Dargestellt sind hier die prozentualen Wechselkursänderungen in einem Monaten (blau) während des gesamten Zeitraumes (die Asienkrise und der Zeitraum danach (49 Muster von Juli 1997 bis Juli 2001) soll prognostiziert werden) gegenüber der Prognose (rot) der Expertenrundentopologie. Auch diese Topologie reagiert zeitverzögert, aber, wie alle anderen Topologien auch, dann mit stärkeren Schwankungen als im übrigen Verlauf. Das heißt: eine Krise wird zwar erkannt, aber zu spät.

5 Fazit und Ausblick

Neuronale Netze bieten eine Möglichkeit Wechselkursprognosen unter der Voraussetzung verschiedenster Modelle zu erstellen. Dieser Aufsatz zeigt anhand eines über die fundamentalen Modelle übergreifenden Ansatzes, wie dies möglich wird und welche Güte die approximierten Lösungen gegenüber dem tatsächlichen Verlauf des Thai Baht-US-Dollar Wechselkurses haben.

Wird beim Training der neuronalen Netze die Asienkrise bzw. auch der Bereich fester Wechselkurse, mit der starken Schwankung des Wechselkurses davor und danach, nicht betrachtet[20], so werden die neuronalen Netze besser trainiert und liefern auf dem Generalisierungsdatensatz bessere Ergebnisse. Allerdings können dann eventuell nicht mehr die Konstellationen der Inputmuster erkannt werden, die zu einer Asienkrise geführt haben oder die bei einem Intervenieren des thailändischen Staates zum Festhalten des Wechselkurses wichtig sind. Wird dagegen der Zeitraum der Asienkrise mittrainiert, dann prognostizieren alle neuronalen Netze auf dem Generalisierungsdatensatz, also in der Zukunft, weitere Krisen, die aber nicht eintreten. Es wäre also möglich neuronale Netze[21] als Indikatoren für Krisen zu benutzen, um „instabile“ Voraussetzungen, die zu einer Krise führen könnten, zu erkennen, jedoch nicht den Anspruch erheben, dass diese Krisen auch tatsächlich eintreten.

Die Untersuchung, ob die Asienkrise von neuronalen Netzen hätte erkannt werden können, liefert nur unzureichende Ergebnisse. Zwar wird eine Krise erkannt, aber meist einen Monat, nachdem sie bereits eingetroffen ist. Dennoch ist, wie schon zuvor erwähnt, festzuhalten, dass die ohne die Asienkrise trainierten neuronalen Netze gute Prognosen für den Thai Baht-US-Dollar Wechselkurs in einem Monat erstellen.

Generell schneiden die besten Prognosen des Wechselkurses in einem bzw. zwei Monaten mit neuronalen Netzen im Vergleich zum Münzwurf[22] besser ab und haben teilweise Trefferquoten von bis zu 60%. Dennoch sind sie alle schlechter als die naive Prognose (Random Walk Modell), wenn gleich mit ihnen auch Gewinn generiert hätte werden können (vgl. Tabelle 3 und 6 U- und ROI-Werte). Der Analyse mit den entsprechenden Gütemaßen ist kritisch gegenüberzustellen, dass der Generalisierungsdatensatz nur aus 51 Mustern besteht. Um eine bessere Aussagekraft der Gütemaße zu erhalten müsste für die Analyse und für das Training eine längere Zeitreihe zur Verfügung stehen[23].

Bei der optischen Analyse ist festzustellen, dass die neuronalen Netze die Struktur des Wechselkurses erlernen, aber teilweise zu früh oder auch zu träge auf Veränderungen reagieren.

Zu untersuchen bleibt in wie weit die einzelnen Einflussgrößen den Wechselkurs bei diesem komplexen Modell bestimmen. Anhand der Struktur der einzelnen Netztopologien ist dies nur schwer zu erklären, da Wechselwirkungen zwischen den 35 Eingabegrößen bestehen können. Dennoch ist dies mit einer Sensitivitätsanalyse zur Ermittlung der relevanten Zusammenhänge möglich (vgl. Grimm (1997), S.43). Dabei werden einzelne Eingabevariablen verändert, während die übrigen festgehalten werden und anschließend deren Auswirkung auf den Wechselkurs beobachtet.

Weiterhin sind Prognosen des Wechselkurses in drei Monaten nicht erfolgt und es muss auch noch untersucht werden, ob die Asienkrise zwei bzw. drei Monate vorher erkannt hätte werden können[24].

---

[20] Bereinigung des Datensatzes von so genannten Ausreißern, die hier zwar der Realität entsprechen, aber das Training eines sonst nicht zu stark schwankenden Wechselkurses behindern.

[21] Auch neuronale Netze ohne Direktverbindungen, da diese auch nur an immer den gleichen Stellen des Generalisierungsdatensatzes starke Schwankungen prognostizieren und sonst eine Gerade in den Daten ausbilden.

[22] Also einer Chance von 50%.

[23] So könnten neuronale Netze trainiert werden, die den Wechselkurs besser prognostizieren. Anzunehmen ist auch, dass die Einstellungen am Neurosimulator FAUN besser an das Problem angepasst werden können, um zu besseren Ergebnissen zu gelangen. Dies bedarf aber längerer Untersuchungen.

[24] Weiter Untersuchungen hätten jedoch den Rahmen dieser Arbeit gesprengt.

Literatur

Borchert M (2003) Geld und Kredit, 8. Aufl., München, Wien

Breitner MH (2003) Nichtlineare, multivariate Approximation mit Perzeptrons und anderen Funktionen auf verschiedenen Hochleistungsrechnern. Akademische Verlagsgesellschaft Aka GmbH, Berlin

Dieckheuer G (2001) Internationale Wirtschaftsbeziehungen, 5.Aufl. München

Finnoff W, Hergert F, Zimmermann HG (1993) Neuronale Netze als Prognosetool in der Ökonomie, in: Siemens AG Arbeitspapier, Zentrale Forschungs- und Entwicklungsabteilung, München

Grimm G (1997) Fundamentale Wechselkursprognose mit Neuronalen Netzen, Traditionelle versus neuere Ansätze zur Wechselkursbestimmung. Gabler, Wiesbaden

Jarchow HJ (2003) Theorie und Politik des Geldes, 11. Aufl. Vandenhoeck & Ruprecht, Göttingen

Jarchow HJ, Rühmann P (2000) Monetäre Außenwirtschaft I Monetäre Außenwirtschaftstheorie, 5. Aufl. Vandenhoeck & Ruprecht, Göttingen

Käsmeier J (1984) Euromärkte und nationale Finanzmärkte: Eine Analyse ihrer Interpedenz, Berlin

MacDonald R, Taylor MP (1992) Exchange Rate Economics. A Survey. „International Monetary Fund Staff Papers“, Vol 39

Mettenheim von HJ (2003) Entwicklung der grob granularen Parallelisierung für den Neurosimulator FAUN 1.0. und Anwendungen in der Wechselkursprognose. Diplomarbeit am Institut für Wirtschaftswissenschaft der Universität Hannover, Königsworther Platz 1, D-30167 Hannover

Müller T, Linder W (2004) Das grosse Buch der technischen Indikatoren, 8.Aufl., Rosenheim

Pentecost EJ (1993) Exchange Rate Dynamics – A Modern Analysis of Exchange Rate Theory and Evidence

Sarno L, Taylor MP (2002) The economics of exchange rates, Cambridge Univ. Press, New York

Anhang A Abbildungen

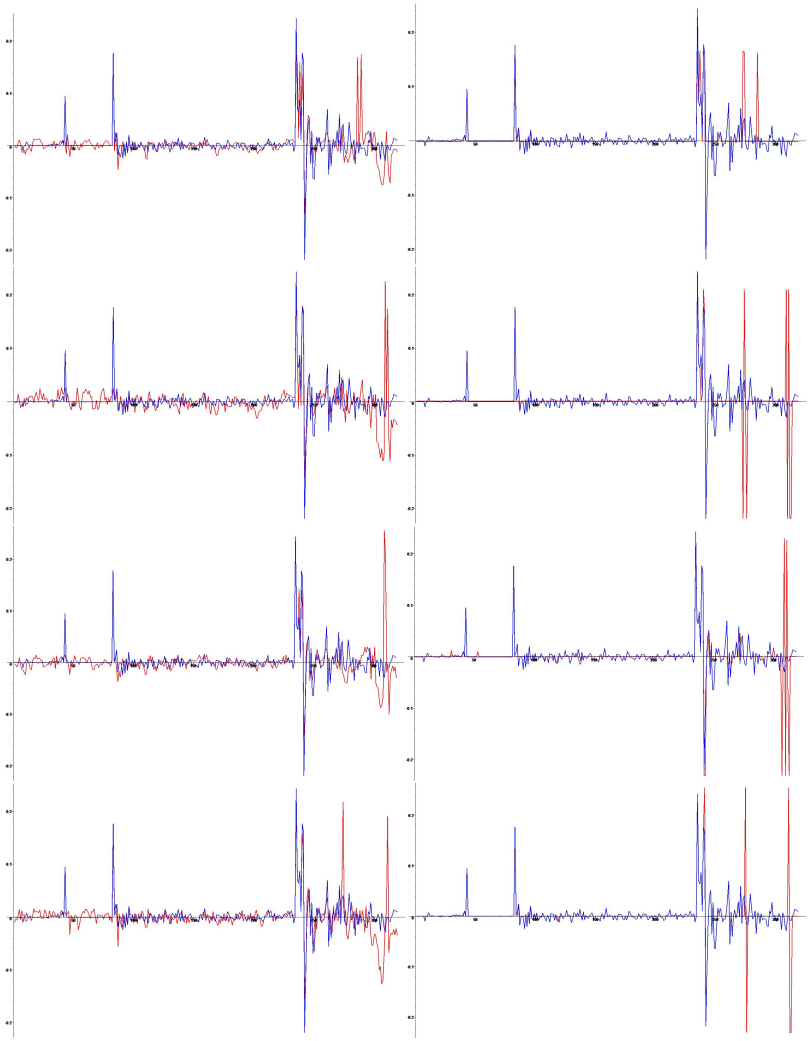


Abb. 10. Vergleich der prozentualen Wechselkursänderungsrate in einem Monat (blau) gegenüber der Prognose (rot) mit einem neuronalen Netz mit einem (erste Zeile) inneren Neuron mit (immer links) und ohne (immer rechts) Direktverbindungen. Zwei, drei und vier inneren Neuronen jeweils zweite Zeile, dritte Zeile und vierte Zeile. Am Ende liegt immer der Generalisierungsdatensatz von 51 Mustern.

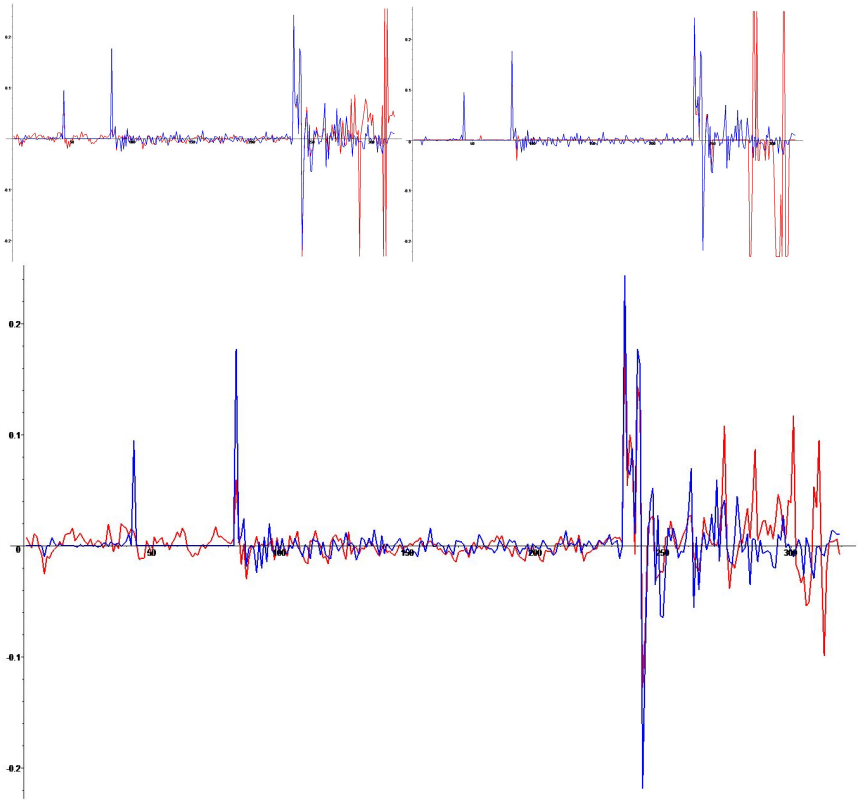


Abb. 11. Vergleich der prozentualen Wechselkursänderungsrate in einem Monat (blau) gegenüber der Prognose (rot) mit einem neuronalen Netz mit fünf inneren Neuron (erste Zeile) mit (links) und ohne (rechts) Direktverbindungen. Unten die Expertenrundentopologie. Am Ende liegt immer der Generalisierungsdatensatz von 51 Mustern.

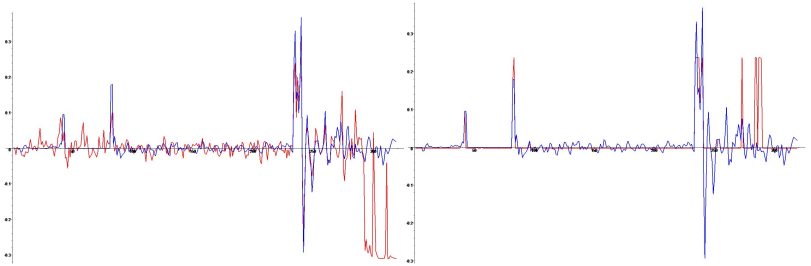


Abb. 12. Vergleich der prozentualen Wechselkursänderungsrate in zwei Monaten (blau) gegenüber der Prognose (rot) mit einem neuronalen Netz mit einem inneren Neuron mit (links) und ohne (rechts) Direktverbindungen. Am Ende liegt immer der Generalisierungsdatensatz von 51 Mustern.

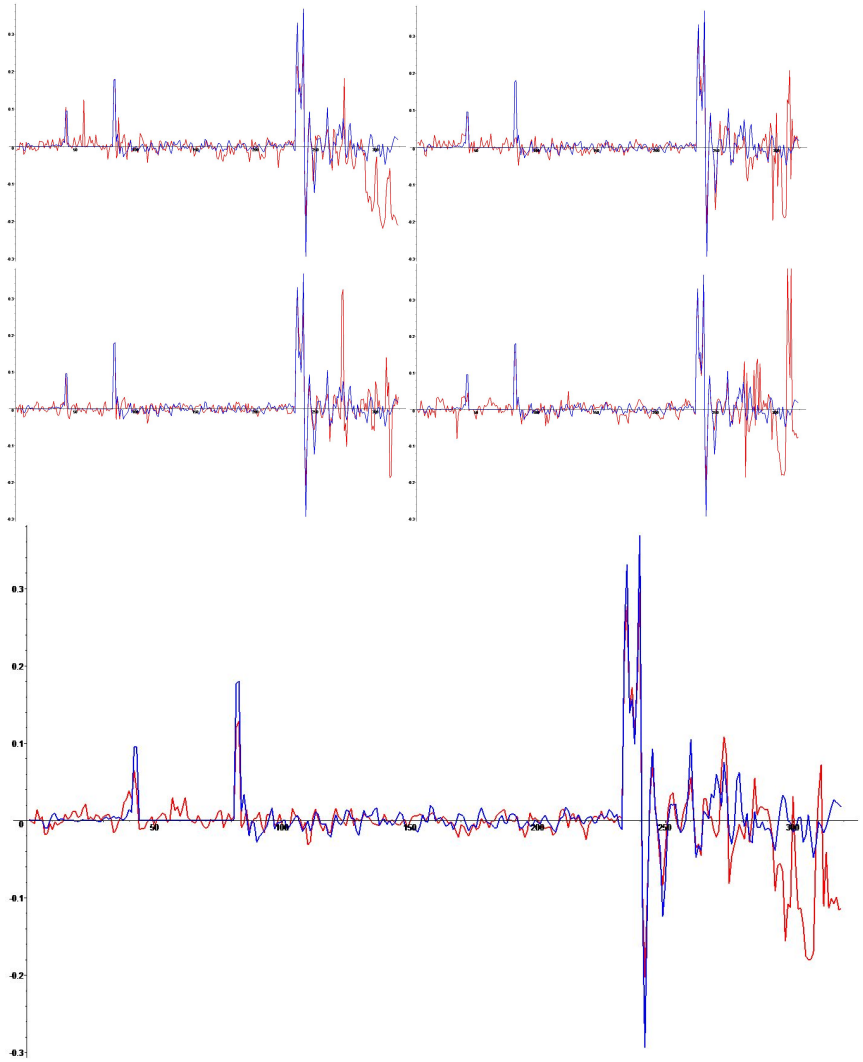


Abb. 13. Vergleich der prozentualen Wechselkursänderungsrate in zwei Monaten (blau) gegenüber der Prognose (rot) mit einem neuronalen Netz mit zwei (erste Zeile links) inneren Neuronen mit Direktverbindungen. Drei (erste Zeile rechts), vier (zweite Zeile links) und fünf (zweite Zeile rechts) inneren Neuronen, jeweils immer mit Direktverbindungen. Unten die Expertenrundentopologie. Am Ende liegt immer der Generalisierungsdatensatz von 51 Mustern.

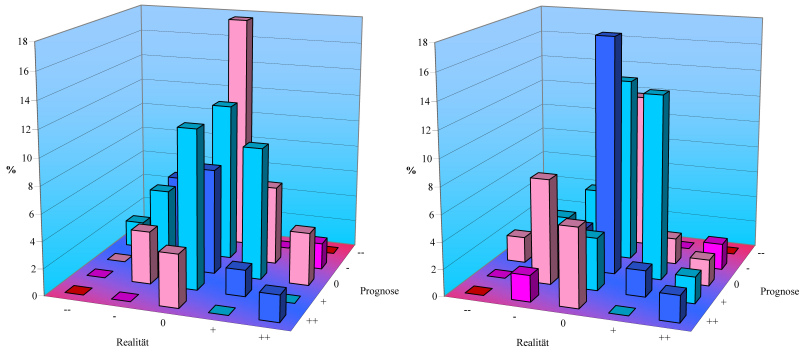


Abb. 14. Prognosehorizont ein Monat: jeweils die beiden besten Netze nach dem Rangsummenverfahren. Links: ein inneres Neuron mit Direktverbindungen, rechts: Expertenrundentopologie.

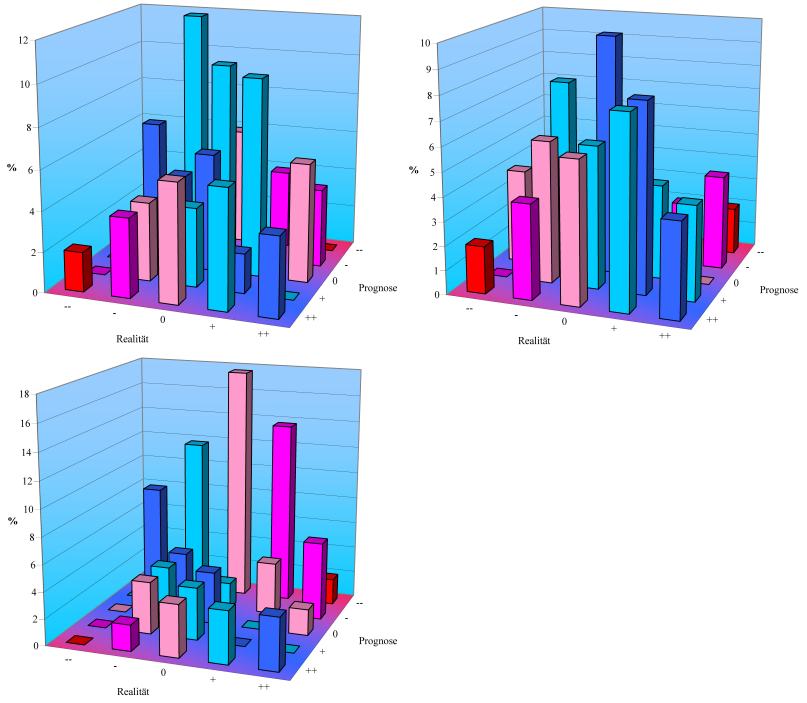


Abb. 15. Prognosehorizont zwei Monate: jeweils die drei besten Netze nach dem Rangsummenverfahren. Links oben: drei innere Neuronen mit Direktverbindungen, rechts oben: vier innere Neuronen mit Direktverbindungen, unten links: Expertenrundentopologie.

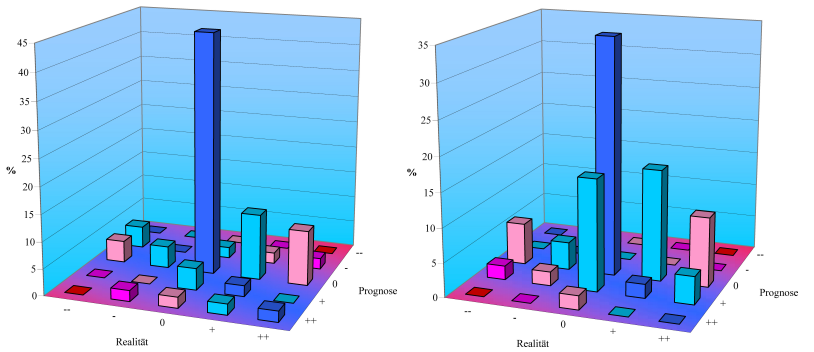


Abb. 16. Prognosehorizont: ein Monat; Training ohne die Asienkrise: jeweils die beiden besten Netze nach dem Rangsummenverfahren. Rechts: ein inneres Neuron mit Direktverbindungen, links: Expertenrundentopologie.

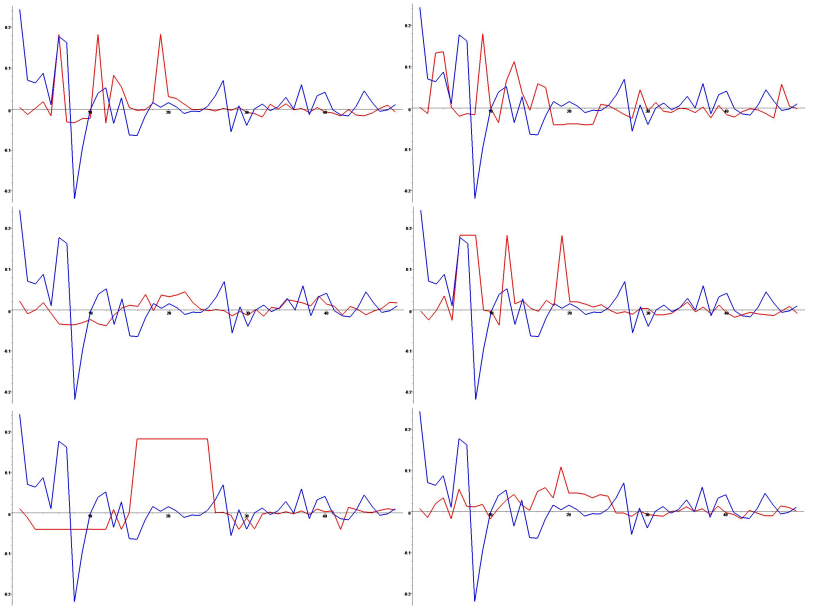


Abb. 17. Hier dargestellt sind die prozentualen Wechselkursänderungen in einem Monaten (blau) während der Asienkrise und der Zeit danach (49 Muster von Juli 1997 bis Juli 2001) gegenüber der Prognose (rot) mit einem neuronalen Netz mit einem (erste Zeile links) inneren Neuronen, zwei (erste Zeile rechts), drei (zweite Zeile links), vier (zweite Zeile rechts), fünf (dritte Zeile links) inneren Neuronen, jeweils immer mit Direktverbindungen. Unten rechts die Expertenrundentopologie. Keine der Topologien sagt den sofortige starken Anstieg des Kurses voraus, sonder reagieren wenn überhaupt erst zeitverzögert.

Anhang B Tabellen

Tabelle 4. Detaillierte Auswertung des Trainings zur Prognose des Wechselkurses in zwei Monaten.

Topologie	I	II	III	IV	V	VI
n_2	1	1	2	3	4	5
Direktverbindungen	nein	ja	ja	ja	ja	ja
Anzahl an Gewichten	38	73	110	147	184	221
ε_i*	0.9126	0.5091	0.5999	0.5125	0.3324	0.3447
$n_i/n_v\,\varepsilon_v$*	0.0436	0.3569	0.2006	0.1467	0.1204	0.2130
durchs. Train. Fehler	0.0917	0.0685	0.0744	0.0687	0.0553	0.0564
proz. Train. Fehler	0.0483	0,0361	0.0391	0.0362	**0.0291**	**0.0297**
Anzahl zufällig initialisierter Netze	536	625	679	761	867	989
Anzahl erfolgreich trainierter Perzzeptrons	500	500	500	500	500	500
% nicht erfolgreich trainierter Netze	6.72%	20%	26.36%	34.3%	42.33%	49.44%
Rechenzeit in sec.	145.5	493.5	835.6	1462.4	2632.1	4434.3

Tabelle 5. Detaillierte Auswertung des Trainings zur Prognose des Wechselkurses in einem Monat ohne die 49 Werte der Asienkrise (219 Trainings- und 51 Validierungsmuster).

Topologie	**I**	**II**	**III**	**IV**	**V**
n_2	1	2	3	4	5
Direktverbindungen	ja	ja	ja	ja	ja
Anzahl an Gewichten	73	110	147	184	221
ε_i*	0.6288	0.4152	1.2879	0.5840	0.8703
$n_i/n_v\,\varepsilon_v$*	0.3475	0.3764	1.2848	0.4301	0.3891
durchs. Train. Fehler	0.0758	0.0616	0.1085	0.0730	0.0892
proz. Train. Fehler	**0,0398**	**0.0324**	0.0571	**0.0384**	0.0469
Anzahl zufällig initialisierter Netze	1511	1499	1660	1632	1644
Anzahl erfolgreich trainierter Perzzeptrons	500	500	500	500	500
% nicht erfolgreich trainierter Netze	66.91%	66.64%	69.88%	69.36%	69.59%
Rechenzeit in sec.	307.8	522.0	935.7	1579.1	1963.8

Tabelle 6. Statistische Gütekriterien für alle berechneten Topologien und der Expertenrundentopologie (Experten) bei der Prognose des Wechselkurses in einem Monat ohne Berücksichtigung der Asienkrise, wobei die Topologie mit einem inneren Neuron und ohne Direktverbindungen (1no) nicht mit in das Rangsummenverfahren eingeht. 1nm = 1 inneres Neuron mit Direktverbindung, usw.

	RMSE		**r**		**WS**		**TQ**		**U**		**ROI**		**RS**	**RANG**
1nm	0.0727	3	0.2319	1	0.4183	1	59.18%	1	0.8156	3	3.3758	3	12	**1**
1no	0.1060	X	-0.0431	X	-0.2116	X	51.02%	X	1.2728	X	-4.1567	X	X	X
2nm	0.0849	5	-0.0739	5	0.0275	5	46.93%	5	0.9858	5	3.3213	4	29	5
3nm	0.0699	1	0.0396	3	0.1141	3	48.98%	4	0.7925	2	3.1981	5	18	**3**
4nm	0.0831	4	0.1029	2	-0.0481	6	51.02%	3	0.9586	4	3.7018	1	20	4
5nm	0.1127	6	-0.1410	6	0.0626	4	46.93%	5	1.3688	6	3.4911	2	29	5
Experten	0.0707	2	0.0223	4	0.1597	2	53.06%	2	0.7895	1	2.7698	6	17	**2**

PROBEN1 — A Set of Neural Network Benchmark Problems and Benchmarking Rules

Lutz Prechelt (prechelt@ira.uka.de)
Fakultät für Informatik
Universität Karlsruhe
76128 Karlsruhe, Germany
++49/721/608-4068, Fax: ++49/721/694092

September 30, 1994

Technical Report 21/94

Abstract

PROBEN1 is a collection of problems for neural network learning in the realm of pattern classification and function approximation plus a set of rules and conventions for carrying out benchmark tests with these or similar problems. PROBEN1 contains 15 data sets from 12 different domains. All datasets represent realistic problems which could be called *diagnosis tasks* and all but one consist of real world data. The datasets are all presented in the same simple format, using an attribute representation that can directly be used for neural network training. Along with the datasets, PROBEN1 defines a set of rules for how to conduct and how to document neural network benchmarking.
The purpose of the problem and rule collection is to give researchers easy access to data for the evaluation of their algorithms and networks and to make direct comparison of the published results feasible. This report describes the datasets and the benchmarking rules. It also gives some basic performance measures indicating the difficulty of the various problems. These measures can be used as baselines for comparison.

Contents

List of Tables

1 Introduction

This section discusses why standardized datasets and benchmarking rules for neural network learning are necessary at all, what the scope of PROBEN1 is, and why real data should be used instead of or in addition to artificial problems as they are often used today.

1.1 Why a benchmark set?

A recent study of the evaluation performed in journal papers about neural network learning algorithms [15] showed that this aspect of neural network research is a rather poor one. Most papers present performance results for the new algorithm only for a very small number of problems — rarely more than three. In most cases, one or several of these problems are meaningless synthetic problems, for instance from the parity/symmetry/encoder family. Comparisons to algorithms suggested by other researchers are in many cases not done at all (exception: standard backpropagation).

Why is this so? Several explanations (read: excuses) are possible:

1. Since training a neural network usually takes quite long, a thorough evaluation takes a very large amount of CPU time.
2. The algorithms of other researchers are often not available as programs at all or their implementations are not stable or are based on some exotic environment.
3. It is difficult to get data for real problems.
4. It is much work to prepare data for neural network training.
5. Even results obtained for the same problem can often not be compared directly because of different problem representations or different experimental setups.

None of these arguments, however, is still a valid one today. I discuss them in order:

1. Not really a problem. Our machines are fast enough now to do a significant amount of training runs within a few days — at least for small or moderately large datasets. And, hey!, it's your computer that must do the work, not you!
2. Yes, often true. And probably nothing that we can easily avoid. However, it would not be a problem if we could just compare against the results of other researchers directly by making the corresponding experiment with a new algorithm.
3. Only partially true. Many researchers who have used real data in their research are willing to give it to others upon request. There are also publicly accessible collections of such data; most notably the UCI machine learning databases repository.
4. Correct. It really is. But not everybody needs to make that data preparation again. We as a research community can and should share the results of such work.
5. This is a real problem that comes in three variants: First, sometimes experimental setups are just plain wrong, giving invalid results. Second, often experimental setups are not documented properly in the papers published, making reproduction or exact comparison impossible. Third, often the documentation just *looks* obscure, because the same things are expressed in very different ways by different people. We need a standard set of conventions for our experiments and their documentation in order to fight this problem.

As we see from this discussion, there is a need for standard sets of problems and rules or conventions for applying them to be used in learning algorithm evaluations. PROBEN1 is meant as a first step towards a set of standard benchmarks for some areas of neural network training algorithm research.

Its availability lays ground for better algorithm evaluations (by enabling easy access to example data of real problems) and for better comparability of the results (if everybody uses the same problems and setup) — while at the same time reducing the workload for the individual researcher.

Aspects of learning algorithms that can be studied using PROBEN1 are for example learning speed, resulting generalization ability, ease of user parameter selection, and network resource consumption. What cannot be assessed well using a set of problems with fixed representation (such as PROBEN1) are all those aspects of learning that have to do with the selection or creation of a suitable problem representation.

Lack of standard problems is widespread in many areas of computer science. At least in some fields, though, standard benchmark problems exist and are used frequently. The most notable of these positive examples are performance evaluation for computing hardware and for compilers. For many other fields it is clear that defining a reasonable set of such standard problems is a very difficult task — but neural network training is not one of them.

1.2 Why benchmarking rules?

It is clear from the discussion above that having a standard set of benchmark problems is, although necessary, not sufficient to improve the de-facto scientific quality of our evaluations. A real improvement is made only if the results published for these benchmark problems are comparable and reproducible. This is not trivial, though, since every application of a neural network training algorithm to a particular problem involves a significant number of user selectable parameters of various kinds — often more than a dozen. If but one of these parameters is not published along with the result, the experiment becomes irreproducible and the comparability of the results is hampered. Even if all parameters are published, comparability might still be an issue due to the fact that many descriptions are ambiguous since we are lacking a standard terminology.

Thus, a set of benchmark problems should be complemented by a set of benchmarking rules (or benchmarking conventions, if you want) that describe and standardize ways of setting up experiments, documenting these setups, measuring results, and documenting these results. Such rules need not reduce the freedom of choosing among several possible experimental setups — they just suggest a core standard that should be used in order to maximize comparability of experimental results and show what should be documented in which way when one deviates from that standard.

As a side-effect, thoroughly documented benchmarking rules reduce the danger that a researcher makes a major fault in his or her experimental setup, thereby producing invalid results.

1.3 Scope of Proben1

Neural network learning algorithm research is a wide field trying to tackle many different classes of problems. Many subfields, such as machine vision, optical character recognition, or speech recognition, are quite specialized and hence also require specialized benchmarks. Other fields require or forbid certain properties to be present in any benchmark problem to be used. Thus, no single set of benchmark problems can be usable for the evaluation of research in the whole field.

The scope of the PROBEN1 problems can be characterized as follows. All problems are suited for supervised learning, since input and output values are separated. All problems are suited for use with networks that do not maintain an internal state, since all examples within a problem are independent of each other. Most of the problems can be tackled by pattern classification algorithms, while a few

others need the capability of continuous multivariate function approximation. Most problems have both continuous and binary input values. All problems are presented as static problems in the sense that all data to learn from is present at once and does not change during learning. All problems except one (the mushroom problem) consist of real data from real problem domains.

The common properties of the learning tasks themselves are characterized by them all being what I call *diagnosis tasks*. Such tasks can be described as follows:

1. The input attributes used are similar to those that a human being would use in order to solve the same problem.
2. The outputs represent either a classification into a small number of understandable classes or the prediction of a small set of understandable quantities.
3. In practice, the same problems *are* in fact often solved by human beings.
4. Examples are expensive to get. This has the consequence that the training sets are not very large.
5. Often some of the attribute values are missing.

The scope of the PROBEN1 rules can be characterized as follows. The rules are meant to apply to all supervised training algorithms. Their presentation, however, is biased towards the training of feed forward networks with gradient descent or similar algorithms. Hence, some of the aspects mentioned in the rules do not apply to all algorithms and some of the aspects relevant to certain algorithms have been left out. The rules suggest certain choices for particular aspects of experimental setups as standard choices and say how to report such choices and the results of the experiments.

Both parts of PROBEN1, problems as well as rules, cover only a small part of neural network learning algorithm research. Additional collections of benchmark problems are needed to cover more domains of learning (e.g. application domains such as vision, speech recognition, character recognition, control, time series prediction; learning paradigms such as reinforcement learning, unsupervised learning; network types such as recurrent networks, analog continuous-time networks, pulse frequency networks. Sufficient benchmarks available today for only a few of these fields). Additions and changes to the rules will also be needed for most of these new domains, learning paradigms, and network types. This is why the digit 1 was included in the name of PROBEN1; maybe some day PROBEN100 will be published and the field will be mature.

1.4 Why no artificial benchmarks?

In the early days of the current era of neural network research (i.e., during the second half of the 1980s), most benchmark problems used were artificial. The most famous one of these is the XOR problem. Its popularity originates from the fact that being able to solve it was the great breakthrough (achieved by the error back-propagation algorithm), compared to the situation faced during the first era of neural network research in the 1960s when no learning algorithm was known to solve a not linearly separable classification task such as XOR.

Other training problems that were often used in the 1980s are the generalized XOR problem (n-bit parity), the n-bit encoder, the symmetry problem, the T-C problem, the 2-clumps problem, and others [3, 18]. Their deficiencies are known: all of these problems are purely synthetic and have strong a-priori regularities in their structure; for some of them it is unclear how to measure in a meaningful way the generalization capabilities of a network with respect to the problem; most of the problems can be solved 100% correct, which is untypical for realistic settings.

Later works used still other synthetic problems which can not be exactly solved so easily. Instances are the two spirals problem [4, 5, 10] or the three discs problem [19]. The problem with these problems

is, similar to the ones mentioned above, that we know a-priori that a simple exact solution exists — at least when using the right framework to express it. It is unclear, how this property influences the observed capability of a learning algorithm or network to find a good solution: some algorithms may be biased towards the kind of regularity needed for a good solution of these problems and will do very good on these benchmarks, although other algorithms not having such bias would be better in more realistic domains.

Summing up, we can conclude that the main problem with the early artificial benchmarks is that we do not know what the results obtained for them tell us about the behavior of our systems on real world tasks.

One way to transcend this limitation is to make the data generation process for the artificial problems resemble realistic phenomena. The usual way to do that is to replace or complement the data generation based on a simple logic or arithmetic formula by stochastic noise processes and/or by realistic models of physical phenomena. Compared to the use of real world data this has the advantage that the properties of each dataset are known, making it easier to characterize for what kinds of problems (i.e., dataset characteristics) a particular algorithm works better or worse than another.

Two problems are left by this approach. First, there is still the danger to prefer algorithms that happen to be biased towards the particular kind of data generation process used. Imagine classification of datasets of point clouds generated by multidimensional gaussian noise using a gaussian-based radial basis function classifier. This can be expected to work very well, since the class of models used by the learning algorithm is exactly the same as the class of models employed in the data generation.[1]

Second, it is often unclear what parameters for the data generation process are representative of real problems in any particular domain. When overlaying a functional and a noise component, the questions to be answered are how strong the non-linear components of the function should be, how strong and of what type the non-linearities in that components should be, and what amount of noise of which type should be added. Choosing the wrong parameters may create a dataset that does not resemble *any* real problem domain.

Clearly artificial datasets based on realistic models and real data sets both have their place in algorithm development and evaluation. A reason for prefering real data over artificially generated data is that the former choice guarantees to get results that are relevant for at least a *few* real domains, namely the ones being tested. Multiple domains must be used in order to increase the confidence that the results obtained did not occur due to a particular domain selection only.

1.5 Related work

Despite the high importance of benchmarks, little is done on the field for neural networks. The only public benchmark collection available that is specifically meant for neural network research is the *Neural Bench* collection at Carnegie Mellon University maintained by Scott Fahlman and collaborators (anonymous ftp to `ftp.cs.cmu.edu`, directory `/afs/cs/project/connect/bench`). Although it was created years ago, it still contains only four sets of data from real world problems.

The only larger collection of benchmark learning problems is the UCI machine learning databases archive (anonymous ftp to `ics.uci.edu`, directory `/pub/machine-learning-databases`). This archive is maintained at the University of California, Irvine, by Patrick M. Murphy and David W. Aha. It contains several dozens of problems, some in multiple variants. The problems in this archive are

[1] My personal impression is that some researchers do this consciously: they make the data generation fit to the known bias of the algorithm they advocate in order to get better results.

meant for general machine learning programs; most of them cannot readily be learned by neural networks because an encoding of nominal attributes and missing attribute values has to be chosen first.

In both collections, the individual datasets themselves were donated by various researchers. With a few exceptions, no partitioning of the dataset into training and test data is defined in the archives. In no case a sub-partitioning of training data into training set and validation set is defined. The different variants that exist for some of the datasets in the UCI archive create a lot of confusion, because it is often not clear which one was used in an experiment. The PROBEN1 benchmark collection contains datasets that are taken from the UCI archive (with one exception). The data is, however, encoded for direct neural network use, is pre-partitioned into training, validation, and test examples, and is presented in a very exactly documented and reproducible form.

Zheng's benchmark [23], which I recommend everybody to read, does not include its own data, but defines a set of 13 problems, predominantly from the UCI archive, to be used as a benchmark collection for classifier learning algorithms. The selection of the problems is made for good coverage of a taxonomy of classification problems with 16 two- or three-valued features, namely type of attributes, number of attributes, number of different nominal attribute values, number of irrelevant attributes, dataset size, dataset density, level of attribute value noise, level of class value noise, frequency of missing values, number of classes, default accuracy, entropy, predictive accuracy, relative accuracy, average information score, relative information score. The PROBEN1 benchmark problems have not explicitly been selected for good coverage of all of these aspects. Nevertheless, for most of the aspects a good diversity of problems is present in the collection.

2 Benchmarking rules

This section describes

- how to conduct valid benchmark tests and
- how to publish them and their results.

The purpose of the rules is to ensure the validity of the results and reproducibility by other researchers. An additional benefit of standardized benchmark setups is that results will more often be directly comparable.

2.1 General principles

The following general principles guide the formulation of the benchmarking rules:

Validity: We need a minimum standard of experimentation that guarantees that the results obtained are valid in the sense that they are not artifacts created by random factors or by a faulty experimental setup. Invalid results are useless. The PROBEN1 benchmarking rules thus contain a number of DOs and DON'Ts to follow in order to avoid invalid results (although following the rules cannot *guarantee* validity of the results).

Reproducibility: The rules prescribe to specify all those aspects of the experimental setup that are needed for other researchers to repeat the experiments. Results that cannot be reproduced are no scientific results. The PROBEN1 benchmarking rules thus attempt to list the relevant aspects of a benchmarking setup that need to be published to attain reproducibility. For many of these aspects, standard formulations are suggested in order to simplify presentation and comprehension.

Comparability: It is very useful if one can compare results obtained by different researchers directly. This is possible if the same experimental setup is used. The rules hence suggest a number of so called *standard choices* for experimental setups that are recommended to be used unless specific reasons stand against it. The use of such standard choices reduces the variability of benchmarking setups and thus improves comparability of results across different publications.

In the rules below, phrases typeset in sans serif font like this indicate suggested formulations to be used in publications in order to reduce the ambiguity of setup descriptions. The following sections present the PROBEN1 benchmarking rules.

2.2 Benchmark problem used

For each benchmark problem X that you use, indicate exactly what X is. In the case of a PROBEN1 problem, just give its name, e.g. hearta. In other cases, specify how and where other researchers can get the problem dataset. Sometimes this can be done by giving a reference to a paper published earlier. Otherwise a file containing the dataset should be available for anonymous FTP somewhere and you should give the FTP address that must be used to get the dataset. If you prepare your own datasets, make them available publicly by FTP if possible. If you use problems from PROBEN1, just cite this report.

Often researchers use a problem that has been used several times before and refer to it by a natural language name, for instance "A test was made using Michalski's soybean data". Such kinds of references often result in confusion, because several different versions of the data exist. So please either refer to a named problem from a well-documented benchmark collection such as PROBEN1 or give the address of a data file available by FTP or reference a paper that does so.

2.3 Training set, validation set, test set

The data used for performing benchmarks on neural network learning algorithms must be split into at least two parts: one part on which the training is performed, called the training data, and another part on which the performance of the resulting network is measured, called the test set. The idea is that the performance of a network on the test set estimates its performance in real use. This means that absolutely no information about the test set examples or the test set performance of the network must be available during the training process; otherwise the benchmark is invalid.

In many cases the training data is further subdivided. Some examples are put into the actual training set, others into a so-called validation set. The latter is used as a pseudo test set in order to evaluate the quality of a network during training. Such an evaluation is called cross validation; it is necessary due to the overfitting (overtraining) phenomenon: For two networks trained on the same problem, the one with larger training set error may actually be *better*, since the other has concentrated on peculiarities of the training set at the cost of losing much of the regularities needed for good generalization [7]. This is a problem in particular when not very many training examples are available.

A popular and very powerful form to use cross validation in neural networks is early stopping: Training proceeds not until a minimum of the error on the training set is reached, but only until a minimum of the error on the validation set is reached during training. Training is stopped at this point and the current network state is the result of the training run. Note that the actual procedure is a bit more

complicated since there may be many local minima in the validation set error curve and since in order to recognize a minimum one has to train until the error rises again, so that resetting the network to an earlier state is needed in order to actually stop at the minimum. See section 3.3 for a more concrete description. Other forms of cross validation besides early stopping are also possible. The data of the validation set could be used in any way during training since it is part of the training data. The actual name 'validation set', however, is only appropriate if the set is used to assess the generalization performance of the network. Note the differentiation: training data is the union of training set and validation set.

Be sure to specify exactly which examples of a dataset are used for the training, validation, and test set. It is insufficient to indicate the number of examples used for each set, because it might make a significant difference *which* ones are used where. As a drastic example think of a binary classification problem where only examples of one class happen to be in the training data.

For PROBEN1, a suggested partitioning into training, validation, and test set is given for each dataset. The size of the training, validation, and test set in all PROBEN1 data files is 50%, 25%, and 25% of all examples, respectively. Note that this percentage information is not sufficient for an exact determination of the sets unless the total number of examples is divisible by four. Hence, the header of each PROBEN1 data file lists explicitly the number of examples to be used for each set. Assume that these numbers are X, Y, and Z. Then the standard partitioning is to use the first X examples for the training set, the following Y examples for the validation set and the final Z examples for the test set. If no validation set is needed, the training set consists of the first $X + Y$ examples instead.

As said before, for problems with only a small number of examples, results may vary significantly for different partitionings (see also the results presented below in section 3.3). Hence it improves the significance of a benchmark result when different partitionings are used during the measurements and results are reported for each partitioning separately. PROBEN1 supports this approach. It contains three different permutations of each dataset. For instance the problem `glass` is available in three datasets `glass1`, `glass2`, and `glass3`, which differ only in the ordering of examples, thereby defining three different partitionings of the `glass` problem data. Additional partitionings (although not completely independent ones) are defined by the following rules for the order of examples in the dataset file:

`a` training set, validation set, test set.
`b` training set, test set, validation set.
`c` validation set, training set, test set.
`d` validation set, test set, training set.
`e` test set, validation set, training set.
`f` test set, training set, validation set.

This list is to be read as follows: From a partitioning, say `glass1`, six partitionings can be created by re-interpreting the data into a different order of training, validation, and test set. For instance `glass1d` means to take the data file of `glass1` and use the first 25% of the examples for the validation set, the next 25% for the test set, and the final 50% for the training set. Obviously, when no validation set is used, `a` is the same as `c` and `e` is the same as `f`, thus only `a`, `b`, `d`, and `e` are available. `glass1a` is identical to `glass1`. The latter is the preferred name when none of `b` to `f` are used in the same context.

Note that these partitionings are of lower quality than those created by the permutations 1 to 3, since the latter are independent of each other while the former are not. Therefore, the additional partitionings should be used only when necessary; in most cases, just using `xx1`, `xx2`, and `xx3` for each problem `xx` will suffice.

If you want to use a different partitioning than these standard ones for a PROBEN1 problem, specify

exactly how many examples for each set you use. If you do not take them from the data file in the order training examples, validation examples, test examples, specify the rule used to determine which examples are in which set. Examples: glass1 with 107 examples used for the training set and 107 examples used for the test set for a standard order but nonstandard size of the sets or glass1 with even-numbered examples used for the training set and odd-numbered examples used for the test set, the first example being number 0 for a nonstandard size and order of sets. If you use the PROBEN1 conventions, just say glass1 and mention somewhere in your article that your benchmarks conform to the PROBEN1 conventions, e.g. All benchmark problems were taken from the Proben1 benchmark set; the standard Proben1 benchmarking rules were applied.

An imprecise specification of the partitioning of a known data set into training, validation and test set is probably the most frequent (and the worst) obstacle to reproducibility and comparability of published neural network learning results.

2.4 Input and output representation

How to represent the input and output attributes of a learning problem in a neural network implementation of the problem is one of the key decisions influencing the quality of the solutions one can obtain. Depending on the kind of problem, there may be several different kinds of attributes that must be represented. For all of these attribute kinds, multiple plausible methods of neural network representation exist. We will now discuss each attribute kind and some common methods to represent such an attribute.

Real-valued attributes are usually rescaled by some function that maps the value into the range $0 \ldots 1$ or $-1 \ldots 1$ in a way that makes a roughly even distribution within that range. They are represented either by a single network input or by a range of inputs using a topological encoding (e.g. overlapping gaussian receptive fields). PROBEN1 always uses a single input for a real-valued attribute, the rescaling function is always linear (with only one exception where the logarithm is used).

Integer-valued attributes are most often handled as if they were real-valued. If the number of different values is only small, one of the representations used for ordinal attributes may also be appropriate. Note that often attributes whose values are integer numbers are not really integer-valued but are ordinal or cardinal instead. PROBEN1 treats all integer-valued attributes as real-valued.

Ordinal attributes with m different values are either mapped onto an equidistant scale making them pseudo-real-valued or are represented by $m - 1$ inputs of which the leftmost k have value 1 to represent the k-th attribute value while all others are 0. A binary code using only $\lceil \log_2 m \rceil$ inputs can also be used. There are only few ordinal attributes in the PROBEN1 problems. For these, either pseudo-real-valued or pseudo-nominal representation is used.

Nominal attributes with m different values are usually either represented using a 1-of-m code or a binary code. With the exception of `gene`, which uses a 2-bit binary code, PROBEN1 always employs 1-of-m representation for nominal attributes.

Missing attribute values can be replaced by a fixed value (e.g. the mean of the non-missing values of this attribute or a value found using an EM algorithm [8]) or can be represented explicitly by adding another input for the attribute that is 1 iff the attribute value is missing. PROBEN1 uses both methods; the fixed value method is used only when but a few of the values are missing. Other methods are possible if one extends the training regime away from static examples, e.g. by using a Boltzmann machine [18].

Most of the above discussion applies to outputs as well, except for the fact that there never are missing outputs. Most PROBEN1 problems are classification problems; all of these are encoded using a 1-of-m output representation for the m classes, even for $m = 2$.

The problem representation in PROBEN1 is fixed. This improves the comparability of results and reduces the work needed run benchmarks. The PROBEN1 datasets are meant to be used exactly as they are. The fixed neural network input and output representation is actually one of the major improvements of PROBEN1 over the previous situation. In the past, most benchmarks consisting of real data were publicly available only in a symbolic representation which can be encoded into a representation suitable for neural networks in many different ways. This fact made comparisons difficult.

When you perform benchmarks that do not use problems from a well-defined benchmark collection, be sure to specify exactly which input and output representation you use. Since such a description consumes a lot of space, the only feasible way will usually be to make the data file used for the actual benchmark runs available publicly.

Should you make small changes to the representation of PROBEN1 problems used in your benchmarks, specify these changes exactly. The most common cases of such changes will be concerned with the output representation. If you want to use only a single output for binary classification problems, say card1, using only one output or something similar. You may also want to ignore one of the outputs for problems having more than two classes, since one output is theoretically redundant since the outputs always sum to one. If you ignore an output, you should always ignore the *last* output from the given representation. If you want to use outputs in the range $-1 \ldots 1$ instead of $0 \ldots 1$ or in a somewhat reduced range in order to avoid saturation of the output nodes, say for example with the target outputs rescaled to the range $-0.9 \ldots 0.9$. It will be assumed that the rescaling was done using a linear transformation of the form $y' = ay + b$. Other possibilities include for instance with the outputs rescaled to mean 0 and standard deviation 1, which will also be assumed to be made using a linear transformation. Of course, all these rescaling modifications can be done for inputs as well, but tell us if you make such changes. I do not recommend to use PROBEN1 problems with representations that differ substantially from the standard ones unless finding good representations an important part of your work.

The input and output representations used in PROBEN1 are certainly not optimal, but they are meant to be good or at least reasonable ones. Differences in problem representation, though, can make for large differences in the performance obtained (see for instance [2]), so be sure to specify your representation precisely.

2.5 Training algorithm

Obviously, an exact specification of the training algorithm used is essential. When you use a known algorithm, specify it by giving a reference to a paper that describes it and then either use the algorithm exactly as specified in that paper or describe precisely all alterations that you make. If you introduce a new algorithm, give the algorithm a name to make it easier for other authors to refer to your algorithm. If there are several variants of your algorithm, give each variant its own name, perhaps by just appending a digit or letter to the primary name.

Whether new algorithm or not, clearly specify the values of all free parameters of the algorithm that you used. When introducing a new algorithm you should clearly indicate a prototype parameter vector (including parameter names) that must be specified to document each use of the algorithm. It is a common error that some of the parameter values used for an algorithm remain unspecified.

These parameters may include (depending on the algorithm) learning rate, momentum, weight decay, initialization, temperature, etc. For each such parameter there should be a clearly indicated unique name and perhaps also a symbol. For all of the parameters that are adaptive, the adaption rule and its parameters have to be specified as well. A particularly important aspect of a training algorithm is its stopping criterion, so do not forget to specify that as well (see section 3.3 for an example).

For all user-selectable parameters, specify how you found the values used and try to characterize how sensitive the algorithm is to their choice. Note that you must not in any way use the performance on the test set while searching for good parameter values; this would invalidate your results! In particular, choosing parameters based on test set results is an error.

2.6 Error measures

Many different error measures (also called error functions, objective functions, cost functions, or loss functions) can be used for network training. The most commonly used is the squared error: $E(o,t) = \sum_i (o_i - t_i)^2$ for actual output values o_i at the i-th output node and target output values t_i for one example. Note that some researchers multiply this by $1/2$ in order to make the derivative simpler[2]; this is considered non-standard. The above measure gives one error value per example — obviously too much data to report. Thus one usually reports either the sum or the average of these values over the set of all examples. The average is called the mean squared error. It has the advantage of being independent of the size of the dataset and is thus preferred. Note that mean squared error still depends on the number of output coefficients in the problem representation and on the range of output values used. I thus suggest to normalize for these factors as well and report a squared error percentage as follows

$$E = 100 \cdot \frac{o_{max} - o_{min}}{N \cdot P} \sum_{p=1}^{P} \sum_{i=1}^{N} (o_{pi} - t_{pi})^2$$

where o_{min} and o_{max} are the minimum and maximum values of output coefficients in the problem representation (assuming these are the same for all output nodes), N is the number of output nodes of the network, and P is the number of patterns (examples) in the data set considered. Note that networks can (and in early training phases often will) produce more than 100% squared error if they use output nodes whose activation is not restricted to the range $o_{min} \ldots o_{max}$.

Other error measures include the softmax error, the cross entropy, the classification figure of merit, linear error, exponential error, minimum variance error, and others [20]. If you use any of these, state the error term explicitly. For some of them, the above idea of error percentages is applicable as well.

The actual target function for classification problems is usually not the continuous error measure used during training, but the classification performance. However, since neural networks with continuous outputs are able to approximate a-posteriori probabilities [16], which are often useful if the network outputs are to be used for further processing steps, the classification performance is not the only measure we are interested in. If space permits, you should thus report the actual error values in addition to the classification performance. Classification performance should be reported in percent of incorrectly classified examples, the classification *error*. This is better than reporting the percentage of correctly classified examples, the classification *accuracy*, because the latter makes important differences insufficiently clear: An accuracy of 98% is actually twice as good as one of 96%, which is easier to see if the errors are reported (2% compared to 4%). If classification accuracy was far below 50% instead of being far above 50%, the accuracy would better be report instead of the error, but this is an

[2] without the factor $1/2$ in the error function, the correct derivative is twice as large as the one that is usually used in formulations of backpropagation. Using the common derivative thus amounts to using halved learning rates.

uncommon case. Avoid the term classification performance, use classification accuracy and classification error instead.

There are several possibilities to determine the classification a network has computed from the outputs of the network. We assume a 1-of-m encoding for m classes using output values 0 and 1. The simplest classification method is the winner-takes-all, i.e., the output with the highest activation designates the class. Other methods involve the possibility of rejection, too. For instance one could require that there is exactly one output that is larger than 0.5, which designates the class if it exists and leads to rejection otherwise. To put an even stronger requirement on the credibility of the network output one can set thresholds, e.g. accept an output as 0 if it is below 0.3 and as 1 if it is above 0.7, and reject unless there is exactly one output that is 1 while all others are 0 by this interpretation. There are several other possibilities. When no rejection capability is needed, the winner-takes-all method is considered standard. In all other cases, describe your classification decision function explicitly.

2.7 Network used

Specify exactly the topology of the neural network used in any benchmark test. The topology of a network is described by the graph of the nodes (units, vertices, neurons) and connections (weights, edges, synapses). Avoid the terms 'neuron' and 'synapse', because they are inappropriate for artificial neural networks. The term 'weight' should be used to refer to the parameter attached to a connection, but not to the connection itself.

To describe the topology, try to refer to common topology models. For instance for the common case of the so-called fully connected layered feed forward networks, the numbers of nodes in each layer from input to output can be given as a sequence: a 5-4-6 network refers to a network with 5 input, 4 hidden, and 6 output nodes. There is confusion how to count the number of layers in a network, so do not call a network like the one above a "three layer network" (counting all groups of nodes) nor a "two layer network" (counting only the groups of nodes with input connections). Instead, call it a network with one hidden layer. This generalizes to arbitrary numbers of layers. For instance, a 5-10-3-5-6 is a three hidden layer network.

Specifying the number of nodes is not sufficient even for the "fully connected" networks, because by this term, some people mean that all connections between adjacent layers are present, while others mean that all connections are present, even those that skip intermediate layers (shortcut connections). Thus, use formulations like with all feed forward connections between adjacent layers or with all feed forward connections, including all shortcut connections as a complement to the specification of the size of the layers. Examples: a 5-4-6 network with all feed forward connections, including all shortcut connections or networks with one hidden layer (having between 2 and 20 hidden nodes) and all feed forward connections between adjacent layers.

Most networks also have a bias (or threshold) for all hidden and output nodes. This bias can be implemented either as an incoming connection from a node with constant non-zero output (the bias node) or as an adaptable parameter of the node activation function. Since the style of implementation is usually irrelevant and networks without bias are the exception, bias need not be mentioned. If some nodes do *not* use bias, specify which (and why). Note that if you compute the number of free parameters in a network, the bias parameter of each hidden and output node has to be included. Since this may confuse the reader, you should mention the bias in this case.

For recurrent networks use standard names such as Jordan or Elman network where appropriate and back it up by a reference or further explanation. Non-standard network topologies or non-standard network models such as networks with shared weights [14] have to be described in detail.

Other properties of the network architecture also have to be specified: the range and resolution of the weight parameters (unless plain 32-bit floating point is used), the activation function of each node in the network (except for the input nodes which are assumed to use the identity function; see also section 2.10), and any other free parameters associated with the network.

2.8 Training results

Usually what one is interested in when training a neural network is its generalization performance. The value that is usually used to characterize generalization performance is the error on a test set. A test set is a set of examples that was not used in any way whatsoever during the training process (see section 2.3 above). This test set error is thus the primary result to be presented for any learning problem used. The corresponding errors on the training and validation set, if any, are of only marginal interest and need thus not be reported.

Since training a neural network usually involves some kind of random initialization, the results of several training runs of the same algorithm on the same dataset will differ. In order to make reliable statements about the performance of an algorithm it is thus necessary to make several runs and report statistics on the distribution of results obtained. If possible, use either 10 runs or 30 runs or some power of ten of these numbers, because if many researchers use the same numbers of runs direct comparisons are easier. If these numbers don't seem appropriate for some reason, try to use either 20 or 60 runs or some power of ten of these numbers. The commonly used statistics to report about the results of the runs should be primarily the mean and $n-1$ standard deviation[3] of test set error (and/or test set classification error) and the 'best' run (see below), secondarily the minimum, maximum, and median, and if still more data shall be presented, all five quartiles or even a fine-grained distribution histogram.

The meaning of the 'best' run result is to characterize what one could get using a method of model selection that trained several networks and then picked that one of them that "looked best". In contrast, all other statistics characterize the quality one can expect if one trains just one network. The selection of the 'best' run must thus not be based on the results of the test set, because that would mean to use the test set error during the model selection process whereas the test set error is conceptually the *result* of the model selection process. Instead, training set error or validation set error or some other quality measure computed exclusively from the network and the training data must be used. This means that the 'best' network will often not have the minimum test error! So for instance if your selection criterion is validation set error, you should report something like the network with lowest validation set error in 30 runs had a test set classification error of 2.34%.

If for some reason you want to exclude some of the runs from the results presented, for instance because these runs are considered to have *not converged* (whatever that may mean), always exclude exactly half of all runs. This allows for easier comparison with the results of other researchers. The runs to be excluded are the worst runs in the inverse sense of 'best' from above, i.e., you must *not* exclude those runs that have the worst test set error.

You may want to apply methods of statistical inference to your training results, for instance in order to test whether one algorithm is *significantly* better than another. In this case, it may be necessary to remove a small number of *outliers* from the samples to be compared in order to make the data satisfy some requirement of the statistical procedure. For instance in order to apply a t-test, the samples to be compared must have a normal distribution. If a sample (of, say, the test errors from 30 runs) is approximately normal except for, say, two outliers with very much larger (or smaller) errors than

[3] That is, standard deviation computed based on the degrees of freedom, which is one less than the number n of runs.

all the rest, you can remove these two outliers from the sample. Never remove more than 10% of the values from any one sample; usually one should remove much less. Never remove an outlier unless the resulting distribution satisfies the requirement well enough. Other data transformations than removing outliers may be more appropriate to satisfy the requirements of the statistical procedure; for instance test errors are often log-normally distributed, so one must use the logarithm of the test error instead of the test error itself in order to produce valid results. See also section 3.3.4.

2.9 Training times

Unless your algorithm does perform a lot of additional work besides propagating data through a neural network, the most sensible measure of training time is the number of connection traversals (connection crossings, sometimes misleadingly called connection updates) needed. This measure is useful because it is independent of a particular machine and implementation. Forward and backward propagation counts individually, for certain algorithms that require more than one quantity to be backwardly propagated through each connection such as [1, 13], each quantity counts as one traversal at each connection. Actual weight update steps also count as one traversal per updated connection. If possible report your training times using the connection traversal measure.

If your algorithm performs much work besides traversing the network, the actual CPU time spent is the best measure to give. The disadvantage of this measure is that it leaves two free parameters: the speed of the machine used and the efficiency of the software implementation. Thus, the measure is directly comparable only for the same software on the same machine. When reporting CPU times, give the precise brand and model number of the machine you used and its nominal performance in SPEC marks; give a hint as to whether the software used should be regarded efficient or not so efficient. However, CPU time is certainly always useful as a ballpark figure for the computational size of the tackled problem.

A less useful measure is the number of *epochs* used, i.e., the number of times each example was processed. This value can be misleading, because the computational cost of one epoch can differ significantly from one algorithm or network to another. It is nevertheless fine to present the epoch counts in addition to other measures.

Regarding non-converging runs [3], the values you report should reflect the actual amount of computation time that was spent. This means that your algorithm should define some stopping or restarting criterion and the sum of all computation actually performed before and after the restart(s) should be reported as the training time. It is important to report the precise stopping or restarting criterion that was used.

2.10 Important details

Finally, some important details are often forgotten; all of them were already shortly mentioned above.

Activation function. Exactly specify the activation function used in the nodes (units) of your network. You can say standard sigmoid to mean $1/(1+e^{-x})$ and you can say tanh to mean the tangens hyperbolicus, which is $2/(1+e^{-2x})-1$; these two are the standard choices. All other activation functions should be given explicitly. Specify whether the output nodes of the network also use this activation function or use the identity function instead. If the nodes of the input layer (fan-out nodes) perform any computation on the input values, specify this computation.

Network initialization. Specify the initialization conditions of the network. The most important point is the initialization of the network's weights, which must be done with random values for most algorithms in order to break the symmetry among the hidden nodes. Common choices for the initialization are for instance fixed methods such as random weights from range $-0.1 \ldots 0.1$, where the distribution is assumed to be even unless stated otherwise, or methods that adapt to the network topology used such as random weights from range $-1/\sqrt{N} \ldots 1/\sqrt{N}$ for connections into nodes with N input connections. Just like the termination criterion, the initialization can have significant impact on the results obtained, so it is important to specify it precisely. Specifying the exact sets of weights used is hopelessly difficult and should usually not be tried.

Termination and phase transition criteria. Specify exactly the criteria used to determine when training should stop, or when training should switch from one phase to the next, if any. For most algorithms, the results are very sensitive to these criteria. Nevertheless, in most publications the criteria are specified only roughly, if at all. This is one of the major weaknesses of many articles on neural network learning algorithms. See section 3.3 for an example of how to report stopping criteria; the GL_α family of stopping criteria, which is defined in that section, is recommended when using the early stopping method.

2.11 Author's quick reference

The following is a quick reference check list of all the points that should be mentioned in a publication reporting a benchmark test. Remember that peculiar points not listed here may apply additionally to the particular benchmarks you want to report.

1. Problem: name, address, version/variant.
2. Training set, validation set, test set.
3. Network: nodes, connections, activation functions.
4. Initialization.
5. Algorithm parameters and parameter adaption rules.
6. Termination, phase transition, and restarting criteria.
7. Error function and its normalization on the results reported.
8. Number of runs, rules for including or excluding runs in results reported.

3 Benchmarking problems

The following subsections each describe one of the problems of the PROBEN1 benchmark set. For each problem, a rough description of the semantics of the dataset is given, plus some information about the size of the dataset, its origin, and special properties, if any. For most of the problems, results have previously been published in the literature. Since these results never use exactly the same representation and training set/test set splitting as the PROBEN1 versions, the references are not given here; some of them can, however, be found in the documentation supplied with the original dataset, which is part of PROBEN1. The final section reports on the results of some learning runs with the PROBEN1 datasets.

3.1 Classification problems

3.1.1 Cancer

Diagnosis of breast cancer. Try to classify a tumor as either benign or malignant based on cell descriptions gathered by microscopic examination. Input attributes are for instance the clump thickness, the uniformity of cell size and cell shape, the amount of marginal adhesion, and the frequency of bare nuclei.

9 inputs, 2 outputs, 699 examples. All inputs are continuous; 65.5% of the examples are benign. This makes for an entropy of 0.93 bits per example[4].

This dataset was created based on the "breast cancer Wisconsin" problem dataset from the UCI repository of machine learning databases. Please mention in any publication presenting results for this data set that the data was originally obtained from the University of Wisconsin Hospitals, Madison, from Dr. William H. Wolberg. Also please cite one or more of the four publications mentioned in the detailed documentation of the original dataset in the `proben1/cancer` directory.

3.1.2 Card

Predict the approval or non-approval of a credit card to a customer. Each example represents a real credit card application and the output describes whether the bank (or similar institution) granted the credit card or not. The meaning of the individual attributes is unexplained for confidence reasons.

51 inputs, 2 outputs 690 examples. This dataset has a good mix of attributes: continuous, nominal with small numbers of values, and nominal with larger numbers of values. There are also a few missing values in 5% of the examples. 44% of the examples are positive; entropy 0.99 bits per example.

This dataset was created based on the "crx" data of the "Credit screening" problem dataset from the UCI repository of machine learning databases.

3.1.3 Diabetes

Diagnose diabetes of Pima indians. Based on personal data (age, number of times pregnant) and the results of medical examinations (e.g. blood pressure, body mass index, result of glucose tolerance test, etc.), try to decide whether a Pima indian individual is diabetes positive or not.

8 inputs, 2 outputs, 768 examples. All inputs are continuous. 65.1% of the examples are diabetes negative; entropy 0.93 bits per example. Although there are no missing values in this dataset according to its documentation, there are several senseless 0 values. These most probably indicate missing data. Nevertheless, we handle this data as if it was real, thereby introducing some errors (or noise, if you want) into the dataset.

This dataset was created based on the "Pima indians diabetes" problem dataset from the UCI repository of machine learning databases.

[4] Entropy $E = \sum_{\text{Classes } c} P(c) \log_2(P(c))$ for class probabilities $P(c)$

3.1.4 Gene

Detect intron/exon boundaries (splice junctions) in nucleotide sequences. From a window of 60 DNA sequence elements (nucleotides) decide whether the middle is either an intron/exon boundary (a donor), or an exon/intron boundary (an acceptor), or none of these.

120 inputs, 3 outputs, 3175 examples. Each nucleotide, which is a four-valued nominal attribute, is enoded binary by two binary inputs (The input values used are -1 and 1, therefore the inputs are *not* declared as boolean. This is the only dataset that has input values not restricted to the range $0 \ldots 1$). There are 25% donors and 25% acceptors in the dataset; entropy 1.5 bits per example.

This dataset was created based on the "splice junction" problem dataset from the UCI repository of machine learning databases.

3.1.5 Glass

Classify glass types. The results of a chemical analysis of glass splinters (percent content of 8 different elements) plus the refractive index are used to classify the sample to be either float processed or non float processed building windows, vehicle windows, containers, tableware, or head lamps. This task is motivated by forensic needs in criminal investigation.

9 inputs, 6 outputs, 214 examples. All inputs are continuous, two of them have hardly any correlation with the result. As the number of examples is quite small, the problem is sensitive to algorithms that waste information. The sizes of the 6 classes are 70, 76, 17, 13, 9, and 29 instances, respectively; entropy 2.18 bits per example.

This dataset was created based on the "glass" problem dataset from the UCI repository of machine learning databases.

3.1.6 Heart

Predict heart disease. Decide whether at least one of four major vessels is reduced in diameter by more than 50%. The binary decision is made based on personal data such as age, sex, smoking habits, subjective patient pain descriptions, and results of various medical examinations such as blood pressure and electro cardiogram results.

35 inputs, 2 outputs, 920 examples. Most of the attributes have missing values, some quite many: For attributes 10, 12, and 11, there are 309, 486, and 611 values missing, respectively. Most other attributes have around 60 missing values. Additional boolean inputs are used to represent the "missingness" of these values. The data is the union of four datasets: from Cleveland Clinic Foundation, Hungarian Institute of Cardiology, V.A. Medical Center Long Beach, and University Hospital Zurich. There is an alternate version of the dataset `heart`, called `heartc`, which contains only the Cleveland data (303 examples). This dataset represents the cleanest part of the heart data; it has only two missing attribute values overall, which makes the "value is missing" inputs of the neural network input representation almost redundant. Furthermore, there are still another two versions of the same data, `hearta` and `heartac`, corresponding to `heart` and `heartc`, respectively. The difference to the datasets described above is the representation of the output. Instead of using two binary outputs to represent the two-class decision "no vessel is reduced" against "at least one vessel is reduced", `hearta` and `heartac` use a single continuous output that represents by the magnitude of its activation the number of vessels that are reduced (zero to four). Thus, these versions of the heart problem are approximation tasks.

The `heart` and `hearta` datasets have 45% patients with "no vessel is reduced" (entropy 0.99 bits per example), for `heartc` and `heartac` the value is 54% (entropy 1.00 bit per example).

These datasets were created based on the "heart disease" problem datasets from the UCI repository of machine learning databases. Note that using these datasets requires to include in any publication of the results the name of the institutions and persons who have collected the data in the first place, namely (1) Hungarian Institute of Cardiology, Budapest; Andras Janosi, M.D., (2) University Hospital, Zurich, Switzerland; William Steinbrunn, M.D., (3) University Hospital, Basel, Switzerland; Matthias Pfisterer, M.D., (4) V.A. Medical Center, Long Beach and Cleveland Clinic Foundation; Robert Detrano, M.D., Ph.D. All four of these should be mentioned for the `heart` and `hearta` datasets, only the last one for the `heartc` and `heartac` datasets. See the detailed documentation of the original datasets in the `proben1/heart` directory.

3.1.7 Horse

Predict the fate of a horse that has a colic. The results of a veterinary examination of a horse having colic are used to predict whether the horse will survive, will die, or will be euthanized.

58 inputs, 3 outputs, 364 examples. In 62% of the examples the horse survived, in 24% it died, and in 14% it was euthanized; entropy 1.32 bits per example. This problem has *very* many missing values (about 30% overall of the original attribute values), which are all represented as missing explicitly using additional inputs.

This dataset was created based on the "horse colic" problem dataset from the UCI repository of machine learning databases.

3.1.8 Mushroom

Discriminate edible from poisonous mushrooms. The decision is made based on a description of the mushroom's shape, color, odor, and habitat.

125 inputs, 2 outputs, 8124 examples. Only one attribute has missing values (30% missing). This dataset is special within the benchmark set in several respects: it is the one with the most inputs, the one with the most examples, the easiest one[5], and it is the only one that is not *real* in the sense that its examples are not actual observations made in the real world, but instead are hypothetical observations based on descriptions of species in a book ("The Audubon Society Field Guide to North American Mushrooms"). The examples correspond to 23 species of gilled mushrooms in the Agaricus and Lepiota Family. In the book, each species is identified as definitely edible, definitely poisonous, or of unknown edibility and not recommended. This latter class was combined with the poisonous one. 52% of the examples are edible (ahem, I mean, have class attribute 'edible'); entropy 1.00 bit per example.

This dataset was created based on the "agaricus lepiota" dataset in the "mushroom" directory from the UCI repository of machine learning databases.

[5] The mushroom dataset is so simple that a net that performs only a linear combination of the inputs can learn it reliably to 0 classification error on the test set!

3.1.9 Soybean

Recognize 19 different diseases of soybeans. The discrimination is done based on a description of the bean (e.g. whether its size and color are normal) and the plant (e.g. the size of spots on the leafs, whether these spots have a halo, whether plant growth is normal, whether roots are rotted) plus information about the history of the plant's life (e.g. whether changes in crop occurred in the last year or last two years, whether seeds were treated, how the environment temperature is).

35 inputs, 19 outputs, 683 examples. This is the problem with the highest number of classes in the benchmark set. Most attributes have a significant number of missing values. The soybean problem has been used often in the machine learning literature, although with several different datasets, making comparisons difficult. Most of the past uses use only 15 of the 19 classes, because the other four have only few instances. In this dataset, these are 8, 14, 15, 16 instances versus 20 for most of the other classes; entropy 3.84 bits per example.

This dataset was created based on the "soybean large" problem dataset from the UCI repository of machine learning databases. Many results for this learning problem have been reported in the literature, but these were based on a large number of different versions of the data.

3.1.10 Thyroid

Diagnose thyroid hyper- or hypofunction. Based on patient query data and patient examination data, the task is to decide whether the patient's thyroid has overfunction, normal function, or underfunction.

21 inputs, 3 outputs, 7200 examples. For various attributes there are missing values which are always encoded using a separate input. Since some results for this dataset using the same encoding are reported in the literature, `thyroid1` is *not* a permutation of the original data, but retains the original order instead. The class probabilities are 5.1%, 92.6%, and 2.3%, respectively; entropy 0.45 bits per example.

This dataset was created based on the "ann" version of the "thyroid disease" problem dataset from the UCI repository of machine learning databases.

3.1.11 Summary

For a quick overview of the classification problems, have a look at table 1. The table summarizes the external aspects of the training problems that you have already seen in the individual descriptions above. It does also discriminate inputs that take on only two different values (binary inputs), inputs that have more than two ("continuous" inputs), and inputs that are present only to indicate that values at some other inputs are missing. In addition, the table indicates the number of attributes of the original problem formulation that were used in the input representation, discriminated to be either binary attributes, "continuous" attributes, or nominal attributes with more than two values.

3.2 Approximation problems

3.2.1 Building

Prediction of energy consumption in a building. Try to predict the hourly consumption of electrical energy, hot water, and cold water, based on the date, time of day, outside temperature, outside air humidity, solar radiation, and wind speed.

Problem	Problem attributes				Input values				Classes	Examples	E
	b	c	n	tot.	b	c	m	tot.	b		
cancer	0	9	0	9	0	9	0	9	2	699	0.93
card	4	6	5	15	40	6	5	51	2	690	0.99
diabetes	0	8	0	8	0	8	0	8	2	768	0.93
gene	0	0	60	60	120	0	0	120	3	3175	1.50
glass	0	9	0	9	0	9	0	9	6	214	2.18
heart	1	6	6	13	18	6	11	35	2	920	0.99
heartc	1	6	6	13	18	6	11	35	2	303	1.00
horse	2	13	5	20	25	14	19	58	3	364	1.32
mushroom	0	0	22	22	125	0	0	125	2	8124	1.00
soybean	16	6	13	35	46	9	27	82	19	683	3.84
thyroid	9	6	0	21	9	6	6	21	3	7200	0.45

Problems and the number of binary, continuous, and nominal attributes in the original dataset, number of binary and continuous network inputs, number of network inputs used to represent missing values, number of classes, number of examples, class entropy E in bits per example. (Continuous means more than two different ordered values).

Table 1: Attribute structure of classification problems

14 inputs, 3 outputs, 4208 examples. This problem is in its original formulation an extrapolation task. Complete hourly data for four consecutive months was given for training, and output data for the following two months should be predicted. The dataset `building1` reflects this formulation of the task: its examples are in chronological order. The other two versions, `building2` and `building3` are random permutations of the examples, simplifying the problem to be an interpolation problem.

The dataset was created based on problem A of "The Great Energy Predictor Shootout — the first building data analysis and prediction problem" contest, organized in 1993 for the ASHRAE meeting in Denver, Colorado.

3.2.2 Flare

Prediction of solar flares. Try to guess the number of solar flares of small, medium, and large size that will happen during the next 24-hour period in a fixed active region of the sun surface. Input values describe previous flare activity and the type and history of the active region.

24 inputs, 3 outputs, 1066 examples. 81% of the examples are zero in all three output values.

This dataset was created based on the "solar flare" problem dataset from the UCI repository of machine learning databases.

3.2.3 Hearta

The analog version of the heart disease diagnosis problem. See section 3.1.6 on page 19 for the description. For `hearta`, 44.7%, 28.8%, 11.8%, 11.6%, 3.0% of all examples have 0, 1, 2, 3, 4 vessels reduced, respectively. For `heartac` these values are 54.1%, 18.2%, 11.9%, 11.6%, and 4.3%.

3.2.4 Summary

For a quick overview of the approximation problems, have a look at table 2. The table summarizes

Problem	Problem attribs.				Input values				Outputs	Examples
	b	c	n	tot.	b	c	m	tot.	c	
building	0	6	0	6	8	6	0	14	3	4208
flare	5	2	3	10	22	2	0	24	3	1066
hearta	1	6	6	13	18	6	11	35	1	920
heartac	1	6	6	13	18	6	11	35	1	303

Problems and the number of binary, continuous, and nominal attributes of the original problem representation used, number of binary and continuous network inputs, number of network inputs used to represent missing values, number of outputs, number of examples. (Continuous means more than two different ordered values).

Table 2: Attribute structure of approximation problems

the external aspects of the training problems that you have already seen in the individual descriptions above. It does also discriminate inputs that take on only two different values (binary inputs), inputs that have more than two ("continuous" inputs), and inputs that are present only to indicate that values at some other inputs are missing. In addition, the table indicates the number of attributes of the original problem formulation that were used in the input representation, discriminated to be either binary attributes, "continuous" attributes, or nominal attributes with more than two values. The outputs have continuous values.

3.3 Some learning results

In this section we will see a few results of neural network learning runs on the datasets described above. The runs were made with linear networks, having only direct connections from the inputs to the outputs, and with various fully connected multi layer perceptrons with one or two layers of sigmoidal hidden nodes.

The method applied for training was the same in all cases and can be summarized as follows: Training was performed using the RPROP algorithm [17] with parameters as indicated below. RPROP is a fast backpropagation variant similar in spirit to Quickprop. It is about as fast as Quickprop but requires less adjustment of the parameters to be stable. The parameters used were not determined by a trial-and-error search, but are just educated guesses instead. RPROP requires epoch learning, i.e., the weights are updated only once per epoch. While epoch updates are is not desirable for very large training sets, it is a good method for small and medium training sets such as those of PROBEN1, because it allows the use of acceleration techniques as those used in RPROP. Conjugate gradient optimization methods would be another class of useful algorithms for this kind of training problems [11].

The squared error function was used. For each dataset, training used the training set and the error on the validation set was measured after every fifth epoch (this interval between two measurements of the validation set error is called the *strip length*, see below). Training was stopped as soon as the GL_5 stopping criterion was fulfilled (see below) or when training progress sank below 0.1 per thousand (see below) or when a maximum of 3000 epochs had been trained. The test set performance was then computed for that state of the network which had minimum validation set error during the training process.

This method, called *early stopping* [6, 9, 12], is a good way to avoid overfitting [7] of the network to the particular training examples used, which would reduce the generalization performance. For optimal performance, the examples of the validation set should be used for further training afterwards, in order not to waste valuable data. Since the optimal stopping point for this additional training is not clear, it was not performed in the experiments reported here.

The GL_5 stopping criterion is defined as follows. Let E be the squared error function. Let $E_{tr}(t)$ be the average error per example over the training set, measured during epoch t. $E_{va}(t)$ is the error on the validation set after epoch t and is used by the stopping criterion. $E_{te}(t)$ is the error on the test set; it is not known to the training algorithm but characterizes the quality of the network resulting from training.

The value $E_{opt}(t)$ is defined to be the lowest validation set error obtained in epochs up to t:

$$E_{opt}(t) = \min_{t' \le t} E_{va}(t')$$

Now we define the *generalization loss* at epoch t to be the relative increase of the validation error over the minimum-so-far (in percent):

$$GL(t) = 100 \cdot \left(\frac{E_{va}(t)}{E_{opt}(t)} - 1 \right)$$

A high generalization loss is one candidate reason to stop training. This leads us to a class of stopping criteria: Stop as soon as the generalization loss exceeds a certain threshold α. We define the class GL_α as

$$GL_\alpha : \text{ stop after first epoch } t \text{ with } GL(t) > \alpha$$

To formalize the notion of training progress, we define a *training strip of length* k to be a sequence of k epochs numbered $n+1 \ldots n+k$ where n is divisible by k. The training *progress* (measured in parts per thousand) measured after such a training strip is then

$$P_k(t) = 1000 \cdot \left(\frac{\sum_{t' \in t-k+1 \ldots t} E_{tr}(t')}{k \cdot \min_{t' \in t-k+1 \ldots t} E_{tr}(t')} - 1 \right)$$

that is, "how much was the average training error during the strip larger than the minimum training error during the strip?" Note that this progress measure is high for instable phases of training, where the training set error goes up instead of down. The progress is, however, guaranteed to approach zero in the long run unless the training is globally unstable (e.g. oscillating). Just like the progress, GL is also evaluated only at the end of each training strip.

3.3.1 Linear networks

A first set of results is shown in the tables 3 (classification problems) and 4 (approximation problems). These tables contain the results of 10 runs training a linear neural network for each of the datasets. The network had no hidden nodes, just direct connections from each input to each output. The output units used the identity activation function, i.e., their output is just the summed input. The RPROP algorithm used the following parameters: $\eta^+ = 1.2$, $\eta^- = 0.5$, $\Delta_0 \in 0.005 \ldots 0.02$ randomly per weight, $\Delta_{max} = 50$, $\Delta_{min} = 0$, initial weights from $-0.01 \ldots 0.01$ randomly. Training was terminated according to the GL_5 stopping criterion using a strip length of 5 epochs.

The results of these training runs give a first impression of how difficult the problems are. There are some interesting observations to be made:

Problem	Training set		Validation set		Test set		Test set classification		Overfit		Total epochs		Relevant epochs	
	mean	stddev	mean	stddev	mean	stddev	mean	stddev	mean	stddev	mean	stddev	mean	stddev
cancer1	4.25	0.00	2.91	0.01	3.52	0.04	2.93	0.18	0.55	0.59	129	13	104	31
cancer2	3.95	0.52	3.77	0.47	4.77	0.39	5.00	0.61	5.36	10.21	87	51	79	51
cancer3	3.30	0.00	4.23	0.04	4.11	0.03	5.17	0.00	0.35	0.64	115	18	92	29
card1	9.82	0.01	8.89	0.11	10.61	0.11	13.37	0.67	4.57	1.05	62	9	26	3
card2	8.24	0.01	10.80	0.16	14.91	0.55	19.24	0.43	4.22	1.08	65	10	23	5
card3	9.47	0.00	8.39	0.07	12.67	0.17	14.42	0.46	1.52	0.69	102	9	44	12
diabetes1	15.39	0.01	16.30	0.04	17.22	0.06	25.83	0.56	0.05	0.07	209	50	203	47
diabetes2	14.93	0.01	17.47	0.02	17.69	0.04	24.69	0.61	0.02	0.02	209	32	204	34
diabetes3	14.78	0.02	18.21	0.04	16.50	0.05	22.92	0.35	0.12	0.17	214	22	185	46
gene1	8.42	0.00	9.58	0.01	9.92	0.01	13.64	0.10	0.03	0.07	47	6	43	10
gene2	8.39	0.00	9.90	0.00	9.51	0.00	12.30	0.14	0.02	0.03	46	4	40	6
gene3	8.21	0.00	9.36	0.01	10.61	0.01	15.41	0.13	0.03	0.06	42	4	39	6
glass1	8.83	0.01	9.70	0.04	9.98	0.10	46.04	2.21	3.81	0.42	129	13	23	5
glass2	8.71	0.09	10.28	0.19	10.34	0.15	55.28	1.27	5.74	0.67	34	6	14	2
glass3	8.71	0.02	9.37	0.06	11.07	0.15	60.57	3.82	1.76	0.57	135	30	27	11
heart1	11.19	0.01	13.28	0.06	14.29	0.05	20.65	0.31	1.14	0.45	134	15	41	5
heart2	11.66	0.01	12.22	0.02	13.52	0.06	16.43	0.40	0.13	0.09	184	14	146	48
heart3	11.11	0.01	10.77	0.02	16.39	0.18	22.65	0.69	0.14	0.23	142	15	113	53
heartc1	10.17	0.01	9.65	0.03	16.12	0.04	19.73	0.56	0.15	0.11	128	10	114	23
heartc2	11.23	0.03	16.51	0.08	6.34	0.25	3.20	1.56	3.98	0.56	136	22	25	10
heartc3	10.48	0.31	13.88	0.33	12.53	0.44	14.27	1.67	6.23	1.15	26	9	12	3
horse1	11.31	0.16	15.53	0.29	12.93	0.38	26.70	1.87	6.22	0.57	27	7	9	2
horse2	8.62	0.28	15.99	0.21	17.43	0.45	34.84	1.38	5.54	0.47	42	16	13	3
horse3	10.43	0.27	15.59	0.30	15.50	0.45	32.42	2.65	6.34	1.07	26	6	8	3
mushroom1	0.014	—	0.014	—	0.011	—	0.00	—	0.00	—	3000	—	3000	—
soybean1	0.65	0.00	0.98	0.00	1.16	0.00	9.47	0.51	0.28	0.18	553	11	418	41
soybean2	0.80	0.00	0.81	0.00	1.05	0.00	4.24	0.25	0.02	0.02	509	19	504	18
soybean3	0.78	0.00	0.96	0.00	1.03	0.00	7.00	0.19	0.03	0.04	533	27	522	28
thyroid1	3.76	0.00	3.78	0.01	3.84	0.01	6.56	0.00	0.01	0.03	104	16	99	22
thyroid2	3.93	0.00	3.55	0.01	3.71	0.01	6.56	0.00	0.01	0.02	98	16	96	16
thyroid3	3.85	0.00	3.39	0.00	4.02	0.00	7.23	0.02	0.02	0.02	114	22	109	21

Training set: mean and standard deviation (stddev) of minimum squared error percentage on training set reached at any time during training.
Validation set: ditto, on validation set.
Test set: mean and stddev of squared test set error percentage at point of minimum validation set error.
Test set classification: mean and stddev of corresponding test set classification error.
Overfit: mean and stddev of GL value at end of training.
Total epochs: mean and stddev of number of epochs trained.
Relevant epochs: mean and stddev of number of epochs until minimum validation error.

Table 3: Linear network results of classification problems

Problem	Training set		Validation set		Test set		Overfit		Total epochs		Relevant epochs	
	mean	stddev	mean	stddev	mean	stddev	mean	stddev	mean	stddev	mean	stddev
building1	0.21	0.01	0.92	0.06	0.78	0.02	2.15	4.64	407	138	401	142
building2	0.34	0.00	0.37	0.00	0.35	0.00	0.00	0.01	298	23	297	23
building3	0.37	0.04	0.38	0.07	0.38	0.08	1.99	4.45	229	107	217	102
flare1	0.37	0.00	0.34	0.01	0.52	0.01	2.17	1.61	41	5	12	4
flare2	0.42	0.00	0.46	0.00	0.31	0.02	0.72	0.90	37	3	16	10
flare3	0.39	0.00	0.46	0.00	0.35	0.00	0.57	0.73	35	6	18	12
hearta1	3.82	0.00	4.42	0.03	4.47	0.06	1.68	0.68	118	12	27	10
hearta2	4.17	0.00	4.28	0.02	4.19	0.01	0.06	0.13	112	10	107	15
hearta3	4.06	0.00	4.14	0.02	4.54	0.01	0.05	0.05	116	8	110	10
heartac1	4.05	0.00	4.70	0.02	2.69	0.02	0.01	0.02	98	10	96	11
heartac2	3.37	0.11	5.21	0.21	3.87	0.16	6.99	2.27	19	4	13	4
heartac3	2.85	0.09	5.66	0.16	5.43	0.23	6.06	0.99	29	9	14	3

(The explanation from table 3 applies, except that the test set classification error data is not present here.)

Table 4: Linear network results of approximation problems

1. Some of the problems seem to be very sensitive to overfitting. They overfit heavily even with only a linear network (e.g. card1, card2, glass1, glass2, heartac2, heartac3, heartc2, heartc3, horse1, horse2, horse3). This suggests that using a cross validation technique such as early stopping is very useful for the PROBEN1 problems.
2. For some problems, there are quite large differences of behavior between the three permutations of the dataset (e.g. test errors of card, heartc, heartac; training times of heartc; overfitting of glass). This illustrates how dangerous it is to compare results for which the splitting of the data into training and test data was not the same.
3. Some of the problems can be solved pretty well with a linear network. So one should be aware that sometimes a 'real' neural network might be an overkill.
4. The mushroom problem is boring. Therefore, only a single run was made. It reached zero test set classification error after only 80 epochs and zero validation set error after 1550 epochs. However, training stopped only because of the 3000 epoch limit; the errors themselves fell and fell and fell. Due to these results, the mushroom problem was excluded from the other experiments. Using the mushroom problem may be interesting, however, if one wants to explore the scaling behavior of an algorithm with respect to the number of available training examples.
5. Some problems exhibit an interesting "inverse" behavior of errors. Their validation error is lower than the minimum training error (e.g. cancer1, cancer2, card1, card3, heart3, heartc1, thyroid2, thyroid3). In a few cases, this even extends to the test error (cancer1, thyroid2).

3.3.2 Choosing multilayer architectures

As a baseline for further comparison, a number of runs was made using multilayer networks with sigmoidal hidden nodes. For each problem, 12 different network topologies were used: one-hidden-layer networks with 2, 4, 8, 16, 24, or 32 hidden nodes and two-hidden-layer networks with 2+2, 4+2, 4+4, 8+4, 8+8, and 16+8 hidden nodes on the first and second hidden layer, respectively. All of these networks had all possible feed forward connections, including all shortcut connections. The sigmoid activation function used was $y = x/(1 + |x|)$.

For each of these topologies, three runs were performed; two with linear output nodes and one with output nodes using the sigmoidal activation function. Note that in this case the sigmoid output nodes perform only a one-sided squashing of the outputs, because the sigmoid range is $-1 \ldots 1$ whereas the target output range is only $0 \ldots 1$. The parameters for the RPROP procedure used in all these runs were $\eta^+ = 1.1$, $\eta^- = 0.5$, $\Delta_0 \in 0.05 \ldots 0.2$ randomly per weight, $\Delta_{max} = 50$, $\Delta_{min} = 0$, initial weights -0.5...0.5 randomly. Exchanging this with the parameter set used for the linear networks would, however, not make much of a difference. Training was stopped when either $P_5(t) < 0.1$ or more than 3000 epochs trained or the following condition was satisfied: The GL_5 stopping criterion was fulfilled at least once *and* the validation error had increased for at least 8 successive strips at least once *and* the quotient $GL(t)/P_5(t)$ had been larger than 3 at least once[6].

The tables in tables 5 and 6 present the topology and results of the network that produced the lowest validation set error of all these runs for each dataset. The tables also contain some indication of the performance of other topologies by giving the number of other runs that were at most 5% (or 10%) worse than the best run, with respect to the validation set error. The range of test set errors obtained for these other topologies is also indicated.

The architectures presented in these tables are probably not the optimal ones, even among those considered in the set of runs presented. Due to the small number of runs per architecture for each problem, a suboptimal architecture has a decent probability of producing the lowest validation set error just by chance. Experience with the early stopping method suggests that using a network considerably larger than necessary often leads to the best results. As a consequence, the architectures presented in the table shown in table 7 were computed from the results of the runs as the suggested architectures for the various datasets to be used for training of fully connected multi layer perceptrons. These architectures are called the *pivot architectures* of the respective problems. The rule for computing which architecture is the pivot architecture uses all runs from the within-5%-of-best category as candidates. From these, the largest architecture is chosen. Should the same largest topology appear among the candidates with both linear and sigmoidal output units, the one with smaller validation set error is chosen, unless the linear architecture appears *twice*, in which case it is preferred regardless of its validation set error. The raw data used for this architecture selection is listed in appendix D.

It should be noted that these pivot architectures are still not necessarily very good. In particular for some of the problems it might be appropriate to train networks without shortcut connections in order to use networks with a much smaller number of parameters. For instance in the glass problems, the shortcut connections amount for as many as 60 weights, which is about the same number as are needed for a complete network using 4 hidden nodes but no shortcut connections. Since the problem has only 107 examples in the training set, it may be a good idea to start without shortcut connections. Similar argumentation applies for several other problems as well. Furthermore, since many of the pivot architectures are one of the two largest architectures available in the selection runs, namely 32+0 or 16+8, networks with still more hidden nodes may produce superior results for some of the problems.

The following section presents results for multiple runs using the pivot architectures, a subsequent section presents results for multiple runs with the same architectures except for the shortcut connections.

3.3.3 Multilayer networks

Tables 8 (classification problems) and 9 (approximation problems) show the results of training with the pivot architectures. For each variant of each problem, 60 runs were performed. The training

[6] The only reason for this complicated criterion is that the same set of runs was also used to investigate the behavior

Problem	Arch		Validation set		Test set		5%	10%	Epochs	Test range
			err	classif.err	err	classif.err				
cancer1	4+2	l	1.53	1.714	1.053	1.149	0	3	75-205	1.176-1.352
cancer2	8+4	l	1.284	1.143	4.013	5.747	0	0	95	—
cancer3	4+4	l	2.679	2.857	2.145	2.299	2	12	55-360	2.112-2.791
card1	4+4	l	8.251	9.827	10.35	13.95	15	23	20-65	10.02-11.02
card2	4+0	l	10.30	10.98	14.88	18.02	6	20	20-50	14.27-16.25
card3	16+8	l	7.236	8.092	13.00	18.02	0	1	50-55	14.52-14.52
diabetes1	2+2	l	15.07	19.79	16.47	25.00	11	23	65-525	16.3-17.52
diabetes2	16+8	l	16.22	21.35	17.46	23.44	4	23	85-335	17.17-18.4
diabetes3	4+4	l	17.26	25.00	15.55	21.35	8	33	65-400	15.65-18.15
gene1	2+0	l	9.708	12.72	10.11	13.37	7	13	30-1245	9.979-11.25
gene2	4+2	s	7.669	11.71	7.967	12.11	0	0	1680	—
gene3	4+2	s	8.371	10.96	9.413	13.62	0	1	1170-1645	9.702-9.702
glass1	8+0	l	8.604	31.48	9.184	32.08	4	21	40-510	8.539-10.32
glass2	32+0	l	9.766	38.89	10.17	52.83	12	31	15-40	9.913-10.66
glass3	16+8	l	8.622	33.33	8.987	33.96	9	35	20-1425	8.795-11.84
heart1	8+0	l	12.58	15.65	14.53	20.00	17	23	40-80	13.64-15.25
heart2	4+0	l	12.02	16.09	13.67	14.78	20	23	25-85	13.03-14.39
heart3	16+8	l	10.27	12.61	16.35	23.91	18	23	40-65	16.25-17.21
heartc1	4+2	l	8.057		16.82	21.33	2	7	35-75	15.60-17.87
heartc2	8+8	l	15.17		5.950	4.000	2	12	15-90	5.257-7.989
heartc3	24+0	l	13.09		12.71	16.00	3	10	10-105	12.95-16.55
horse1	4+0	l	15.02	28.57	13.38	26.37	8	29	15-45	12.7-14.97
horse2	4+4	l	15.92	30.77	17.96	38.46	21	34	15-45	16.55-19.85
horse3	8+4	l	15.52	29.67	15.81	29.67	14	31	15-35	15.63-17.88
soybean1	16+8	l	0.6715	4.094	0.9111	8.824	0	0	1045	—
soybean2	32+0	l	0.5512	2.924	0.7509	4.706	0	1	895-2185	0.8051-0.8051
soybean3	16+0	l	0.7147	4.678	0.9341	7.647	0	3	565-945	0.9539-0.9809
thyroid1	16+8	l	0.7933	1.167	1.152	2.000	1	1	480-1170	1.194-1.194
thyroid2	8+4	l	0.6174	1.000	0.7113	1.278	0	0	2280	—
thyroid3	16+8	l	0.7998	1.278	0.8712	1.500	2	3	590-2055	0.9349-1.1

Arch: nodes in first hidden layer + nodes in second hidden layer, sigmoidal or linear output nodes for 'best' network, i.e., network used in the run with lowest validation set error.
Validation set: squared error percentage on validation set, classification error on validation set of 'best' run (missing values are due to technical-historical reasons).
Test set: squared error percentage on test set, classification error on test set of 'best' run.
5%: number of *other* runs with validation squared error at most 5 percent worse than that of best run (as shown in second column).
10%: ditto, at most 10% worse.
Epochs: Range of number of epochs trained for best run and within-10-percent-best runs.
Test range: Range of squared test set error percentages for within-10-percent-best runs excluding the 'best' run.

Table 5: Architecture finding results of classification problems

Problem	Arch		Validation set	Test set	5%	10%	Epochs	Test range
building1	2+2	l	0.7583	0.6450	4	13	625-2625	0.6371-0.6841
building2	16+8	s	0.2629	0.2509	5	20	1100-2960	0.246-0.2731
building3	8+8	s	0.2460	0.2475	5	16	600-2995	0.2526-0.2739
flare1	4+0	s	0.3349	0.5283	10	30	35-160	0.5232-0.5687
flare2	2+0	s	0.4587	0.3214	14	30	35-135	0.3167-0.3695
flare3	2+0	l	0.4541	0.3568	14	32	40-155	0.3493-0.3772
hearta1	32+0	s	4.199	4.428	19	33	35-180	4.249-4.733
hearta2	2+0	l	3.940	4.164	3	23	20-120	3.948-4.527
hearta3	4+0	s	3.928	4.961	15	31	20-105	4.337-5.089
heartac1	2+0	l	4.174	2.665	0	3	50-95	2.613-3.328
heartac2	8+4	l	4.589	4.514	0	2	15	4.346-4.741
heartac3	4+4	l	5.031	5.904	8	16	10-55	4.825-6.540

(The explanation from table 5 applies, except that the test set classification error data is not present here.)

Table 6: Architecture finding results of approximation problems

Problem	pivot arch		w	Problem	pivot arch		w	Problem	pivot arch		w
building1	16+0	l	333	building2	16+8	l	605	building3	16+8	s	605
cancer1	4+2	l	100	cancer2	8+4	l	196	cancer3	16+8	l	436
card1	32+0	l	1832	card2	24+0	l	1400	card3	16+8	l	1528
diabetes1	32+0	l	370	diabetes2	16+8	l	410	diabetes3	32+0	l	370
flare1	32+0	s	971	flare2	32+0	s	971	flare3	24+0	s	747
gene1	4+2	l	1115	gene2	4+2	s	1115	gene3	4+2	s	1115
glass1	16+8	l	572	glass2	16+8	l	572	glass3	16+8	l	572
heart1	32+0	l	1288	heart2	32+0	l	1288	heart3	32+0	l	1288
hearta1	32+0	l	1220	hearta2	16+0	l	628	hearta3	32+0	l	1220
heartac1	2+0	l	110	heartac2	8+4	l	512	heartac3	16+8	s	1052
heartc1	16+8	l	1112	heartc2	8+8	l	744	heartc3	32+0	l	1288
horse1	16+8	l	1793	horse2	16+8	l	1793	horse3	32+0	l	2161
soybean1	16+8	l	4153	soybean2	32+0	l	4841	soybean3	16+0	l	3209
thyroid1	16+8	l	794	thyroid2	8+4	l	398	thyroid3	16+8	l	794

Pivot architecture and the corresponding number w of connections for each data set.

Table 7: Pivot architectures for the datasets

parameters used are the same as for the linear networks as indicated in section 3.3.1. Several interesting observations can be made (please compare also with the discussion of the linear network results in section 3.3.1):

1. The results for some of the problems are *worse* than those obtained using linear networks. This is most notable for the gene problems and less severe for the horse problems and many of the heart disease problems.
2. Not surprisingly, the standard deviations of validation and test set errors and the the tendency to overfit are much higher than for linear networks in most of the cases.
3. The correlation of validation set errors with test set errors is quite small for some of the problems (less than 0.5 for cancer3, card3, flare3, glass1, heartac1, heartc1, horse1, horse2, soybean3). In

of different stopping criteria. Those results, however, are not reported here.

Problem	Training set		Validation set		Test set		ρ	Test set classification		Overfit		Total epochs		Relevant epochs	
	mean	stddev	mean	stddev	mean	stddev		mean	stddev	mean	stddev	mean	stddev	mean	stddev
cancer1	2.87	0.27	1.96	0.25	1.60	0.41	0.81	1.47	0.60	4.48	4.87	152	111	133	97
cancer2	2.08	0.35	1.77	0.32	3.40	0.33	0.51	4.52	0.70	5.76	6.70	93	75	81	72
cancer3	1.73	0.19	2.86	0.11	2.57	0.24	0.28	3.37	0.71	3.37	1.32	66	20	51	16
card1	8.92	0.54	8.89	0.59	10.53	0.57	0.92	13.64	0.85	3.77	4.47	33	7	25	5
card2	7.12	0.55	11.11	0.32	15.47	0.75	0.53	19.23	0.80	3.32	1.03	32	8	22	6
card3	7.58	0.87	8.42	0.37	13.03	0.50	-0.03	17.36	1.61	3.52	1.46	37	10	28	9
diabetes1	14.74	2.03	16.36	2.14	17.30	1.91	0.99	24.57	3.53	2.31	0.67	196	98	118	72
diabetes2	13.12	1.35	17.10	0.91	18.20	1.08	0.77	25.91	2.50	2.75	2.54	119	42	85	31
diabetes3	13.34	1.11	17.98	0.62	16.68	0.67	0.55	23.06	1.91	2.34	0.65	307	193	200	132
gene1	6.45	0.42	10.27	0.31	10.72	0.31	0.76	15.05	0.89	2.67	0.49	46	9	29	6
gene2	7.56	1.81	11.80	1.19	11.39	1.28	0.97	15.59	1.83	2.12	0.44	321	698	222	595
gene3	6.88	1.76	11.18	1.06	12.14	0.95	0.95	17.79	1.73	2.06	0.50	435	637	289	508
glass1	7.68	0.79	9.48	0.24	9.75	0.41	0.33	39.03	8.14	2.76	0.71	67	44	45	39
glass2	8.43	0.53	10.44	0.48	10.27	0.40	0.72	55.60	2.83	4.27	1.75	29	9	20	7
glass3	7.56	0.98	9.23	0.25	10.91	0.48	0.54	59.25	7.83	2.68	0.47	66	46	45	41
heart1	9.25	1.07	13.22	1.32	14.33	1.26	0.97	19.89	2.27	2.83	1.89	65	16	43	12
heart2	9.85	1.68	13.06	3.29	14.43	3.29	0.98	17.88	1.57	3.27	2.34	57	19	38	13
heart3	9.43	0.64	10.71	0.78	16.58	0.39	0.67	23.43	1.29	3.35	3.72	51	10	37	9
heartc1	6.82	1.20	8.75	0.71	17.18	0.79	0.10	21.13	1.49	4.04	2.98	45	12	36	11
heartc2	10.41	1.76	17.02	1.12	6.47	2.86	0.83	5.07	3.37	4.05	1.89	29	14	21	11
heartc3	10.30	1.79	15.17	1.83	14.57	2.82	0.85	15.93	2.93	8.22	18.67	24	13	17	11
horse1	9.91	1.06	16.52	0.67	13.95	0.60	0.30	26.65	2.52	4.66	2.28	28	5	20	4
horse2	7.32	1.52	16.76	0.64	18.99	1.21	0.30	36.89	2.12	3.87	1.49	31	8	22	8
horse3	9.25	2.36	17.25	2.41	17.79	2.45	0.92	34.60	2.84	3.48	1.26	30	10	21	7
soybean1	0.32	0.08	0.85	0.07	1.03	0.05	0.54	9.06	0.80	2.55	1.37	665	259	551	218
soybean2	0.42	0.06	0.67	0.06	0.90	0.08	0.77	5.84	0.87	2.17	0.16	792	281	675	243
soybean3	0.40	0.07	0.82	0.06	1.05	0.09	0.33	7.27	1.16	2.16	0.13	759	233	639	205
thyroid1	0.60	0.53	1.04	0.61	1.31	0.55	0.99	2.32	0.67	3.06	3.16	491	319	432	266
thyroid2	0.59	0.24	0.88	0.19	1.02	0.18	0.85	1.86	0.41	2.58	1.07	660	460	598	417
thyroid3	0.69	0.20	0.97	0.13	1.16	0.16	0.91	2.09	0.31	2.39	0.43	598	624	531	564

Training set: mean and standard deviation (stddev) of minimum squared error percentage on training set reached at any time during training.
Validation set: ditto, on validation set.
Test set: mean and stddev of squared test set error percentage at point of minimum validation set error.
ρ: Correlation between validation set error and test set error.
Test set classification: mean and stddev of corresponding test set classification error.
Overfit: mean and stddev of GL value at end of training.
Total epochs: mean and stddev of number of epochs trained.
Relevant epochs: mean and stddev of number of epochs until minimum validation error.

Table 8: Pivot architecture results of classification problems

Problem	Training set		Validation set		Test set		ρ	Overfit		Total epochs		Relevant epochs	
	mean	stddev	mean	stddev	mean	stddev		mean	stddev	mean	stddev	mean	stddev
building1	0.63	0.50	2.43	1.50	1.70	1.01	0.96	31.93	44.07	394	602	329	529
building2	0.23	0.02	0.28	0.02	0.26	0.02	0.98	0.11	0.70	1183	302	1175	303
building3	0.22	0.02	0.26	0.01	0.26	0.01	0.93	0.42	1.09	1540	466	1408	505
flare1	0.39	0.26	0.55	0.81	0.74	0.80	1.00	3.13	2.48	71	28	52	21
flare2	0.42	0.16	0.55	0.43	0.41	0.47	1.00	3.20	3.73	60	15	42	10
flare3	0.36	0.01	0.49	0.01	0.37	0.01	0.32	2.58	0.58	76	28	51	18
hearta1	3.75	0.76	4.58	0.81	4.76	1.14	0.95	4.98	7.85	46	16	34	13
hearta2	3.69	0.87	4.47	1.00	4.52	1.10	0.97	7.18	24.23	59	21	45	19
hearta3	3.84	0.66	4.29	0.73	4.81	0.87	0.97	5.34	14.19	45	13	35	13
heartac1	3.86	0.32	4.87	0.23	2.82	0.22	-0.06	3.98	2.25	44	23	34	21
heartac2	3.41	0.42	5.51	0.65	4.54	0.87	0.79	7.53	5.27	22	9	16	7
heartac3	2.23	0.57	5.38	0.37	5.37	0.56	0.80	4.64	2.96	38	10	30	10

(The explanation from table 8 applies, except that the test set classification error data is not present here.)

Table 9: Pivot architecture results of approximation problems

two cases it is even slightly negative (card3, heartac1).

4. The correlation value also differs dramatically between the three variants of some of the problems (card, flare, heartac, heartc, horse).
5. However, low correlation does not necessary imply bad overall test error results (see cancer, card, flare, heartac, horse).
6. The training times exhibit dramatic fluctuations in a few of the cases (building1, gene2, gene3, thyroid3, and less severely cancer1, cancer2, diabetes3, glass1, glass3, thyroid1, thyroid2).
7. The other numbers of training epochs tend to be of the same order as for linear networks, with a few exceptions that are much faster (most of the heart disease problems) or much slower (thyroid, building2, building3).
8. The "inverse" error behavior observed for some of the linear networks is no longer present for most of them (except cancer1, cancer2, card1).

As mentioned above, for some of the problems it might be more appropriate to work without shortcut connections. To quantify the effect of training without shortcut connections, another series of 60 runs per dataset was conducted using the same parameters as above. This time, however, the network architecture used was modified to include only connections between adjacent layers, i.e., no direct connections from the inputs to the outputs and for networks with two hidden layers also no connections from the inputs to the second hidden layer and from the first hidden layer to the outputs. I call these architectures the no-shortcut architectures.

The results of these runs are shown in tables 10 (classification problems) and 11 (approximation problems). Once again, a few interesting observations can be made (compare also with the above discussions of linear network and pivot architecture results):

1. Leaving out the shortcut connections seems to be appropriate more often than expected (see also section 3.3.4).
2. The test error results for the gene problems are better than for linear networks (for pivot architectures they were worse than the for linear networks). However, the classification errors are worse even than for the pivot architectures.

Problem	Training set		Validation set		Test set		ρ	Test set classification		Overfit		Total epochs		Relevant epochs	
	mean	stddev	mean	stddev	mean	stddev		mean	stddev	mean	stddev	mean	stddev	mean	stddev
cancer1	2.83	0.15	1.89	0.12	1.32	0.13	0.64	1.38	0.49	3.10	2.54	116	123	95	115
cancer2	2.14	0.23	1.76	0.14	3.47	0.28	0.14	4.77	0.94	3.82	1.90	54	31	44	28
cancer3	1.83	0.26	2.83	0.13	2.60	0.22	0.59	3.70	0.52	3.33	1.64	54	20	41	17
card1	8.86	0.41	8.69	0.26	10.35	0.29	0.25	14.05	1.03	3.54	1.25	30	7	22	5
card2	7.18	0.51	10.87	0.27	14.94	0.64	0.44	18.91	0.86	3.99	1.52	26	7	17	5
card3	7.13	0.62	8.62	0.46	13.47	0.51	0.41	18.84	1.19	4.81	3.24	29	7	22	6
diabetes1	14.36	1.14	15.93	1.04	16.99	0.91	0.95	24.10	1.91	2.23	0.53	201	119	117	83
diabetes2	13.04	1.27	16.94	0.91	18.43	1.00	0.76	26.42	2.26	2.50	0.50	102	46	70	26
diabetes3	13.52	1.46	17.89	0.90	16.48	1.16	0.91	22.59	2.23	2.32	0.59	251	132	164	85
gene1	2.70	1.52	8.19	1.33	8.66	1.28	0.91	16.67	3.75	2.46	0.53	124	58	101	53
gene2	4.55	2.60	9.46	1.95	9.54	1.91	0.97	18.41	6.93	2.29	0.28	321	284	250	255
gene3	4.99	2.79	9.45	2.17	10.84	1.93	0.97	21.82	7.53	2.33	0.39	262	183	199	163
glass1	7.16	0.65	9.15	0.21	9.24	0.32	0.13	32.70	5.34	2.69	0.64	71	31	52	27
glass2	8.42	0.66	10.03	0.27	10.09	0.28	0.37	55.57	3.70	4.00	1.80	30	9	22	8
glass3	7.54	1.06	9.14	0.24	10.74	0.52	0.73	58.40	7.82	2.97	1.17	60	30	46	26
heart1	9.24	0.82	13.10	0.65	14.19	0.64	0.89	19.72	0.96	3.16	2.38	57	15	38	12
heart2	9.73	1.24	12.32	1.09	13.61	0.89	0.88	17.52	1.14	3.56	3.47	51	15	36	12
heart3	9.46	0.88	10.85	1.39	16.79	0.77	0.93	24.08	1.12	3.91	4.42	46	13	32	10
heartc1	5.98	1.33	8.08	0.49	16.99	0.77	0.22	20.82	1.47	5.08	2.64	38	10	30	9
heartc2	9.85	1.16	16.86	0.70	5.05	1.36	0.40	5.13	1.63	4.83	2.34	25	10	18	9
heartc3	10.35	1.07	14.30	1.21	13.79	2.62	0.75	15.40	3.20	9.73	10.48	17	6	11	5
horse1	10.43	1.23	15.47	0.37	13.32	0.48	0.24	29.19	2.62	6.09	2.53	19	3	13	3
horse2	6.68	1.85	16.07	0.79	17.68	1.41	-0.19	35.86	2.46	4.28	1.67	25	7	18	6
horse3	10.54	1.68	15.91	1.19	15.86	1.17	0.88	34.16	2.32	5.51	3.89	20	5	14	5
soybean1	1.53	0.09	1.94	0.06	2.10	0.07	0.58	29.40	2.50	3.14	1.99	219	112	159	79
soybean2	0.46	0.19	0.59	0.13	0.79	0.22	0.96	5.14	1.05	5.06	6.49	417	222	362	202
soybean3	0.61	0.21	0.93	0.21	1.25	0.15	0.76	11.54	2.32	6.12	7.99	450	273	382	228
thyroid1	0.59	0.20	1.01	0.16	1.28	0.12	0.84	2.38	0.35	3.99	7.14	377	308	341	280
thyroid2	0.60	0.13	0.89	0.11	1.02	0.11	0.59	1.91	0.24	4.71	6.86	421	269	388	246
thyroid3	0.74	0.18	0.98	0.13	1.26	0.14	0.92	2.27	0.32	3.91	9.18	324	234	298	223

(The explanation from table 8 applies)

Table 10: No-shortcut architecture results of classification problems

3. The test error results for the horse problems have also improved, yet are still worse than for linear networks.

4. The correlations of validation and test error are sometimes very different than for the pivot architectures (see for example card, flare, glass, heartac).

5. For flare2 and flare3, although the correlation is much lower, the standard deviations of test errors are very much smaller, compared to pivot architectures.

Problem	Training set		Validation set		Test set		ρ	Overfit		Total epochs		Relevant epochs	
	mean	stddev	mean	stddev	mean	stddev		mean	stddev	mean	stddev	mean	stddev
building1	0.47	0.28	2.07	1.04	1.36	0.63	0.88	33.93	49.93	307	544	248	457
building2	0.24	0.15	0.30	0.19	0.28	0.20	1.00	0.14	0.78	1074	338	1044	330
building3	0.22	0.01	0.26	0.01	0.26	0.01	0.74	0.25	0.58	1380	350	1304	360
flare1	0.35	0.02	0.35	0.01	0.54	0.01	0.10	3.02	0.90	48	20	35	16
flare2	0.40	0.01	0.47	0.01	0.32	0.01	0.43	2.93	0.99	47	11	32	8
flare3	0.37	0.01	0.47	0.01	0.36	0.01	0.34	2.53	0.47	57	21	32	11
hearta1	3.55	0.53	4.48	0.35	4.55	0.41	0.93	4.17	7.53	47	18	35	16
hearta2	3.45	0.56	4.41	0.21	4.33	0.15	0.55	2.91	0.75	54	22	41	20
hearta3	3.74	0.72	4.46	1.01	4.89	0.91	0.99	5.35	9.90	46	17	34	15
heartac1	3.59	0.24	4.77	0.32	2.47	0.38	0.21	3.78	1.85	42	22	32	18
heartac2	2.58	0.42	5.16	0.32	4.41	0.56	-0.15	6.43	4.43	24	7	18	7
heartac3	2.45	0.46	5.74	0.36	5.55	0.52	0.84	5.52	4.02	31	12	23	10

(The explanation from table 8 applies, except that the test set classification error data is not present here.)

Table 11: No-shortcut architecture results of approximation problems

3.3.4 Comparison of multilayer results

Table 12 shows a comparison of the pivot architecture and no-shortcut architecture results presented above. The comparison was performed with the `ttest` procedure of the SAS statistical software

Problem	1	2	3
building	(—)	N 2.1	P 7.8
cancer	N 0.0	—	—
card	N 0.1	N 0.0	P 0.0
diabetes	—	P 2.9	N 2.1
flare	N 0.0	N 0.0	N 0.0
gene	N 0.0	(N 0.0)	(N 0.0)
glass	N 0.0	N 0.1	N 3.2
heart	N 1.6	N 0.6	—
hearta	—	—	P 0.0
heartac	N 0.0	—	(P 0.0)
heartc	—	N 0.0	N 2.6
horse	N 0.0	N 0.0	N 0.0
soybean	P 0.0	(N 0.0)	P 0.0
thyroid	P 6.9	—	P 0.1

Results of statistical significance test performed for differences of mean logarithmic test error between pivot architectures (P) and no-shortcut architectures (N). Entries show differences that are significant on a 90% confidence level plus the corresponding p-value (in percent); the letter indicates which architecture is better. Dashes indicate non-significant differences. Parentheses indicate unreliable test results due to non-normality of at least one of the two samples. The test employed was a t-test using the Cochran/Cox approximation for the unequal variance case. 2.6% of the data points were removed as outliers.

Table 12: t-test comparison of pivot and no-shortcut results

package. Since a t-test assumes that the samples to be compared have normal distributions, the logarithm of the test errors was compared instead of the test errors themselves, because test errors usually have an approximately log-normal distribution. This logarithmic transformation does not change the test result, since the logarithm is strictly monotone; log-normal distributions occur quite often and log-transformations are a very common statistical technique. Since a further assumption of the t-test is equal variance of the samples, the Cochran/Cox approximation for the unequal variance

case had to be used, because at least for some of the sample pairs (cancer1, gene1, hearta1) the standard deviations differed by more than factor 2. Furthermore, a few outliers had to be removed in order to achieve an approximate normal distribution of the log-errors: In the 2520 runs for the pivot architectures, there were 4 outliers with too low errors and 61 with too high errors. For the no-shortcut architectures, there were no outliers with too low errors and 66 outliers with too high errors. Altogether this makes for 2.6% of outliers. At most 10% outliers, i.e., 6 of 60, were removed from any single sample, namely from heartac2 and heartc3 (pivot) and from horse3 (no-shortcut). A few of the samples deviated so significantly from a log-normal distribution that the results of the test are unreliable and thus must be interpreted with care. For the pivot architectures, these non-normal samples were those of building1, gene2, and gene3, for the no-shortcut architectures they were building1, gene3, heartac3, and soybean2. No outliers were removed from the non-normal samples. The respective test results are shown in parentheses in the table in order to indicate that they are unreliable. This discussion demonstrates how important it is to work very carefully when applying statistical methods to neural network training results. When applied carelessly, statistical methods can produce results that may look very impressive but in fact are just garbage.

For 10 of the sample pairs no significant difference of test set errors is found at the 90% confidence level (i.e., significance level 0.1). In 9 cases the pivot architecture was better while in 23 cases the no-shortcut architecture was better. This result suggests that further search for a good network architecture may be worthwhile for most of the problems, since the architectures used here were all found using candidate architectures with shortcut connections only and just removing the shortcut connections is probably not the best way to improve on them.

Summing up, the network architectures and performance figures presented above provide a starting point for exploration and comparison using the PROBEN1 benchmark collection datasets. It must be noted that none of the above results used the validation set for training. Surely, improvements of the results are possible by using the validation set for training in a suitable way. The properties of the benchmark problems seem to be diverse enough to make PROBEN1 a useful basis for improved experimental evaluation of neural network learning algorithms. Hopefully, many more similar collections will follow.

A Availability of Proben1, Acknowledgements

The PROBEN1 benchmark set (including this report) is available for anonymous FTP from the Neural Bench archive[7] at Carnegie Mellon University (machine `ftp.cs.cmu.edu`, directory `/afs/cs/project/connect/bench/contrib/prechelt`) and from machine `ftp.ira.uka.de` in directory `/pub/neuron`. The file name in both cases is `proben1.tar.gz`. This file contains the complete directory tree, including all data, documentation, and the techreport. The size of the file is about 2 MB[8]. When unpacked, the PROBEN1 benchmark set needs about 20 MB disk space. Of these, the actual data files consume about 15 MB.

The present report alone is available for anonymous FTP from machine `ftp.ira.uka.de` in directory `/pub/papers/techreports/1994` as file `1994-21.ps.Z`.

The original datasets on which the PROBEN1 datasets are based are included in the tar files. Their sources are the UCI machine learning databases repository and the energy predictor shootout archive. The UCI machine learning databases repository is available by anonymous FTP on machine `ics.uci.edu` in directory `/pub/machine-learning-databases`. This archive is maintained at the University of California, Irvine, by Patrick M. Murphy and David W. Aha. Many thanks to them for their valuable service. The databases themselves were donated by various researchers; thanks to them as well. See the documentation files in the individual dataset directories for details. The building problem is from the energy predictor shootout archive at `ftp.cs.colorado.edu` in directory `/pub/distribs/energy-shootout`.

If you publish an article about work that used PROBEN1, it would be great if you dropped me a note with the reference to prechelt@ira.uka.de.

[7] Maintained by Scott Fahlman and collaborators. Many thanks to them for their service.

[8] The file is a GNU `gzip`'ed Unix tar format file. The GNU `gzip` compression utility is needed to uncompress it.

B Structure of the Proben1 directory tree

The PROBEN1 directory tree that results from unpacking the archive file has the following structure. The top directory is called `proben1`; it contains a `README` file for a quick overview, a `Doc` subdirectory, a `Scripts` subdirectory, and one subdirectory per problem, named like the problem itself.

The `Doc` directory contains this report as both a TEX `dvi` file and as a Postscript file.

The `Scripts` directory contains a number of small Perl scripts that I have used during the preparation of the datasets. I include them in the PROBEN1 distribution for all those people who want to generate additional datasets in the PROBEN1 file format or who want to change the representation used in one of the original PROBEN1 problems. These scripts are not needed for normal use of the PROBEN1 datasets.

Each problem subdirectory for a problem `xx` contains the following files: `README` gives an overview of the files in the directory plus a short description of the attribute encoding used in the PROBEN1 representation of the problem compared to the original representation. `xx1.dt, xx2.dt`, and `xx3.dt` are the actual data files. The only difference between them is that the examples are in a different order (which is always a random permutation, except for `building1` and `thyroid1`). `raw2cod` is the Perl script that was used to convert the original data file into the PROBEN1 data file. This script is the definitive documentation of the problem representation used (with respect to the original data). The problems `heart, hearta, heartac,` and `heartc` are all in the directory `heart`.

C Proben1 file format and data encoding

The following is what a data file looks like (example from glass1.dt):

```
bool_in=0
real_in=9
bool_out=6
real_out=0
training_examples=107
validation_examples=54
test_examples=53
0.281387 0.36391 0.804009 0.23676 0.643527 0.0917874 0.261152 0 0 1 0 0 0 0 0
0.260755 0.341353 0.772829 0.46729 0.545966 0.10628 0.255576 0 0 0 1 0 0 0 0
[further data lines deleted]
```

Each line after the header lines represents one example; first the examples of the training set, then those of the validation set, then those of the test set. The sizes of these sets are given in the last three header lines (the partitioning is always 50%/25%/25% of the total number of examples). The first four header lines describe the number of input coefficients and output coefficients per example. A boolean coefficient is always represented as either 0 (false) or 1 (true). A real coefficient is represented as a decimal number between 0 and 1. For all datasets, either bool_in or real_in is 0 and either bool_out or real_out is 0. Coefficients are separated by one or multiple spaces; examples (including the last) are terminated by a single newline character. First on each line are the input coefficients, then follow the output coefficients (i.e., each line contains `bool_in` + `real_in` + `bool_out` + `real_out` coefficients). Thus, lines can be quite long.
That's all.

The encoding used in the data files has all inputs and outputs scaled to the range 0...1. The scaling was chosen so that the range is at least almost (but not always completely) used by the examples occurring in the dataset. The `gene` datasets are an exception in that they use binary inputs encoded as -1 and 1.

D Architecture ordering

The following list gives for each problem the order of architectures according to increasing squared test set error. Of all architectures tried (as listed in section 3.3.2) only those are listed whose test set error was at most 5% larger than the smallest test set error found in any run. Architectures with linear output units can occur twice, because two runs were made for them and each run is considered separately.

building1: 2+2l, 4+2l, 4+4l, 4+0l, 16+0l.
building2: 16+8s, 16+8l, 8+4s, 16+8l, 32+0s, 32+0l.
building3: 8+8s, 32+0s, 4+4l, 32+0l, 16+8s, 8+4s.
cancer1: 4+2l.
cancer2: 8+4l.
cancer3: 4+4l, 16+8l, 16+0l.
card1: 4+4l, 8+8l, 8+8l, 16+0l, 8+4l, 8+0l, 16+8l, 4+2l, 4+4l, 32+0l, 4+2l, 8+4l, 16+0l, 2+0l, 4+0l, 24+0l.
card2: 4+0l, 16+0l, 16+8l, 24+0l, 2+2l, 8+4l, 4+4l.
card3: 16+8l.
diabetes1: 2+2l, 2+2l, 4+4l, 4+4l, 32+0l, 8+4l, 16+0l, 4+0l, 16+8l, 8+0l, 24+0l, 16+0l.
diabetes2: 16+8l, 24+0l, 8+0l, 8+4l, 4+4l.
diabetes3: 4+4l, 8+0l, 32+0l, 24+0l, 8+8l, 24+0s, 2+2l, 32+0l, 8+8l.
flare1: 4+0s, 2+2l, 4+0l, 2+2s, 2+0l, 32+0s, 4+2l, 2+2l, 2+0s, 4+0l, 16+0s.
flare2: 2+0s, 4+0s, 8+0s, 8+8s, 16+0s, 2+2s, 24+0s, 2+0l, 4+0l, 32+0s, 4+0l, 2+0l, 4+2s, 4+4l, 8+4s.
flare3: 2+0l, 2+0l, 2+0s, 4+0s, 2+2s, 2+2l, 2+2l, 4+4s, 16+0s, 16+0l, 4+0l, 4+4l, 8+0l, 24+0s, 8+8l.
gene1: 2+0l, 2+0s, 4+0l, 2+2l, 2+0l, 2+2l, 4+0l, 4+2l.
gene2: 4+2s.
gene3: 4+2s.
glass1: 8+0l, 16+8l, 4+0l, 32+0s, 8+4l.
glass2: 32+0l, 2+2s, 16+0s, 32+0s, 2+0l, 16+8l, 4+4s, 8+0s, 16+8s, 4+0s, 16+8l, 16+0l, 2+0s.
glass3: 16+8l, 2+0s, 16+0l, 16+0l, 8+4s, 16+8s, 8+8s, 8+4l, 2+0l, 16+8l.

heart1: 8+0l, 24+0l, 4+0l, 32+0l, 16+8l, 8+4l, 32+0l, 8+8l, 16+0l, 4+2l, 4+0l, 4+4l, 8+4l, 24+0l, 2+2l, 4+4l, 8+0l, 16+8l.
heart2: 4+0l, 16+0l, 32+0l, 4+0l, 8+0l, 32+0l, 4+4l, 8+8l, 2+2l, 8+4l, 16+8l, 2+0l, 2+2l, 24+0l, 2+0l, 4+2l, 4+2l, 16+0l, 24+0l, 8+8l, 4+4l.
heart3: 16+8l, 2+0l, 32+0l, 4+0l, 8+0l, 2+2l, 32+0l, 8+8l, 16+0l, 4+2l, 4+4l, 16+0l, 16+8l, 4+4l, 8+4l, 8+4l, 4+2l, 24+0l, 24+0l.
hearta1: 32+0s, 8+4l, 4+0l, 4+4s, 32+0l, 2+0l, 8+0l, 8+4l, 8+8l, 16+8l, 8+0l, 4+0l, 2+2l, 32+0l, 24+0l, 4+2l, 4+4l, 16+0l, 16+8s, 8+8s.
hearta2: 2+0l, 8+4l, 16+0l, 4+2s.
hearta3: 4+0s, 16+0l, 4+4l, 4+0l, 8+0l, 4+4l, 8+4l, 8+0l, 2+0l, 32+0l, 4+2s, 24+0l, 8+8l, 16+8l, 4+2l, 16+8l.
heartac1: 2+0l.
heartac2: 8+4l.
heartac3: 4+4l, 8+4s, 8+0l, 4+0l, 24+0l, 16+8s, 4+0s, 16+0s, 8+0s.
heartc1: 4+2l, 8+8l, 16+8l.
heartc2: 8+8l, 2+2l, 4+0l.
heartc3: 24+0l, 32+0l, 8+8l, 16+8l.
horse1: 4+0l, 4+4l, 4+4l, 4+2s, 2+2l, 8+0s, 16+8l, 4+0s, 16+0l.
horse2: 4+4l, 4+0l, 8+0s, 2+2s, 8+4l, 4+4s, 8+0l, 2+2l, 8+8l, 4+4l, 2+0l, 4+2l, 16+0l, 16+8l, 8+0l, 16+8l, 2+0s, 16+8s, 4+0l, 8+8s, 4+2s, 8+4s.
horse3: 8+4l, 8+0l, 8+4s, 4+0l, 4+4s, 2+0l, 8+8l, 16+0l, 4+4l, 4+4l, 32+0l, 24+0s, 4+2s, 8+0l, 16+8l.
soybean1: 16+8l.
soybean2: 32+0l.
soybean3: 16+0l.
thyroid1: 16+8l, 8+8l.
thyroid2: 8+4l.
thyroid3: 16+8l, 8+4l, 8+8l.

References

[1] Yann Le Cun, John S. Denker, and Sara A. Solla. Optimal brain damage. In *[22]*, pages 598–605, 1990.

[2] T.G. Dietterich and G. Bakiri. Error-correcting output codes: A general method for improving multiclass inductive learning programs. In *Proc. of the 9th National Conference of Artificial Intelligence (AAAI)*, pages 572–577, Anaheim, CA, 1991. AAAI Press.

[3] Scott E. Fahlman. An empirical study of learning speed in back-propagation networks. Technical Report CMU-CS-88-162, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, September 1988.

[4] Scott E. Fahlman and Christian Lebiere. The cascade-correlation learning architecture. Technical Report CMU-CS-90-100, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, February 1990.

[5] Scott E. Fahlman and Christian Lebiere. The Cascade-Correlation learning architecture. In *[22]*, pages 524–532, 1990.

[6] William Finnoff, Ferdinand Hergert, and Hans Georg Zimmermann. Improving model selection by nonconvergent methods. *Neural Networks*, 6:771–783, 1993.

[7] Stuart Geman, Elie Bienenstock, and René Doursat. Neural networks and the bias/variance dilemma. *Neural Computation*, 4:1–58, 1992.

[8] Michael I. Jordan and Robert A. Jacobs. Hierarchical mixtures of experts and the EM algorithm. *Neural Computation*, 6:181–214, 1994.

[9] K.J. Lang, A.H. Waibel, and G.E. Hinton. A time-delay neural network architecture for isolated word recognition. *Neural Networks*, 3(1):33–43, 1990.

[10] K.J. Lang and M.J. Witbrock. Learning to tell two spirals apart. In *Proc. of the 1988 Connectionist Summer School*. Morgan Kaufmann, 1988.

[11] Martin Møller. A scaled conjugate gradient algorithm for fast supervised learning. *Neural Networks*, 6(4):525–533, June 1993.

[12] N. Morgan and H. Bourlard. Generalization and parameter estimation in feedforward nets: Some experiments. In *[22]*, pages 630–637, 1990.

[13] Michael C. Mozer and Paul Smolensky. Skeletonization: A technique for trimming the fat from a network via relevance assessment. In *[21]*, pages 107–115, 1989.

[14] Steven J. Nowlan and Geoffry E. Hinton. Simplifying neural networks by soft weight-sharing. *Neural Computation*, 4(4):473–493, 1992.

[15] Lutz Prechelt. A study of experimental evaluations of neural network learning algorithms: Current research practice. Technical Report 19/94, Fakultät für Informatik, Universität Karlsruhe, D-76128 Karlsruhe, Germany, August 1994. Anonymous FTP: /pub/papers/techreports/1994/1994-19.ps.Z on ftp.ira.uka.de.

[16] Michael D. Richard and Richard P. Lippmann. Neural network classifiers estimate bayesian a-posteriori probabilities. *Neural Computation*, 3:461–483, 1991.

[17] Martin Riedmiller and Heinrich Braun. A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In *Proceedings of the IEEE International Conference on Neural Networks*, San Francisco, CA, April 1993. IEEE.

[18] David Rumelhart and John McClelland, editors. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume Volume 1. MIT Press, Cambridge, MA, 1986.

[19] Steen Sjøgaard. *A Conceptual Approach to Generalisation in Dynamic Neural Networks*. PhD thesis, Aarhus University, Aarhus, Danmark, 1991.

[20] Brian A. Telfer and Harold H. Szu. Energy functions for minimizing misclassification error with minimum-complexity networks. *Neural Networks*, 7(5):809–818, 1994.

[21] David S. Touretzky, editor. *Advances in Neural Information Processing Systems 1*, San Mateo, California, 1989. Morgan Kaufman Publishers Inc.

[22] David S. Touretzky, editor. *Advances in Neural Information Processing Systems 2*, San Mateo, California, 1990. Morgan Kaufman Publishers Inc.

[23] Zijian Zheng. A benchmark for classifier learning. Technical Report TR474, Basser Department of Computer Science, University of Sydney, N.S.W Australia 2006, November 1993. anonymous ftp from ftp.cs.su.oz.au in /pub/tr.

Digital Analog Converter

A digital analog converter (DAC) translates analog signals into digital values and vice versa. This functionality can be implemented by artificial neural networks. DACs are particularly useful in long-distance data transmissions and for controlling machine tools.

The digital type of representation of signals or of data is a type of representation by individual physical dimension, e. g. by a number system. An analog signal is characterized by a constantly variable physical dimension. In this work digital signals are represented by the dual number system and analog signals are represented by the decimal number system. These systems belong to the positional systems. Here the digits and their placing in a number play an important role: Nominal value is the value of a digit, place value is the value of the place, where the digit occurs in a number. With the structure of a notational system N numbers of a character set are given. The character set of the binary system, which represents the digital signals, consists of the digits "0" and "1". The ith digit of a number X is called x_i. Therefore x_i is the nominal value and N^{i-1} is the place value. The value of a n-digit number x with digits $x_n, x_{n-1}, \ldots, x_1$ is

$$x = \sum_{i=1}^{n} x_i N^{i-1}$$

The following example shows a dual number with four digits and its value in the decimal system:

$$0011 = 0 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 3$$

Digital signals are coded as dual numbers. Usually two successive values can differ in more than one bit (see table **??**). Although training of neural networks may succeed, in general using a coding where two successive values differ in *only* one bit simplifies the training of neural networks, so chances reducing the training error to zero increase. A possible alignment is shown in figure **??**.

Another way to generate an appropriate alignment is to use Gray code. Designated after Frank Gray, who patented Gray code in 1953, this coding is used particularly in coding of genetic algorithms. The usage of Gray code is quite straightforward. Be X a dual number as vector of digits, consisting of n digits:

$$X = (x_1, x_2, \ldots, x_n)$$

The Gray code is represented by $G = (g_1, g_2, \ldots, g_n)$, where holds:

$$\begin{aligned} g_1 &= x_1 \\ g_k &= x_{k-1} \oplus x_k, \ k = 2, 3, \ldots, n \end{aligned}$$

The symbol $\oplus$ stands for a direct sum. The transformation between decimal, dual and Gray code is shown in table **??**.

This alignment simplifies the training of neural networks. Dependent from the direction of the training the input and output layer of the neural networks have different meanings:

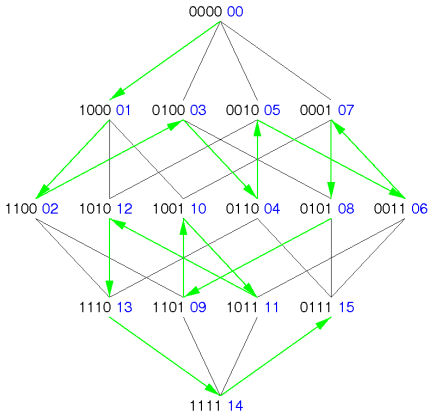


Figure 1: A possible alignment of dual numbers where only one bit changes between two successive values. Blue numbers show the corresponding analog values.

Decimal	Dual	Gray
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100

Decimal	Dual	Gray
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

Table 1: Relationship between decimal and dual numbers and the Gray code

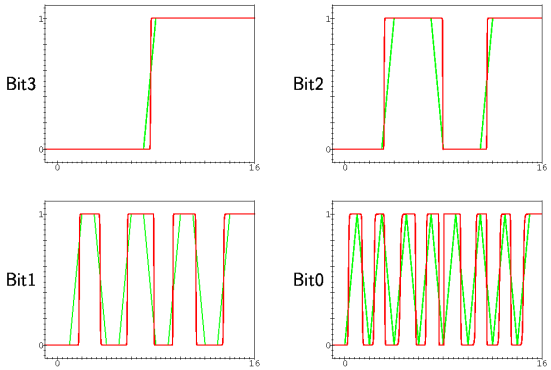


Figure 2: Neural networks trained separately for each bit. Green lines show the target output at $0, 1, 2, \ldots, 15$. The red lines visualize the artificial neural network output. Note that this picture shows the toggling of *dual numbers* instead of the modified alignment using Gray code.

Analog to digital One input neuron and four output neurons. However, computing four neural networks, i. e. one net for each bit, would probably be a more promising approach. Figure **??** shows four neural networks as red lines that have been trained separately for each bit. It can be made out very easy how each net toggles between "0" and "1" and vice versa.

Digital to analog Four input neurons and one output neuron.

In order to simulate a robust DAC with neural networks noisy patterns, e. g. 14.1 in addition to 14, can be trained.

It is possible to find neural networks with a training error of zero for a 4-digit DAC. However, a neural network converting numbers consisting of eight bits reliably has not been found yet.

IWI Discussion Paper Series

Michael H. Breitner, *Rufus Philip Isaacs and the Early Years of Differential Games*, 36 p., #1, January 22, 2003.

Gabriela Hoppe and Michael H. Breitner, *Classification and Sustainability Analysis of E-Learning Applications*, 26 p., # 2, February 13, 2003.

Tobias Brüggemann and Michael H. Breitner, *Preisvergleichsdienste: Alternative Konzepte und Geschäftsmodelle*, 22 p., # 3, February 14, 2003.

Patrick Bartels and Michael H. Breitner, *Automatic Extraction of Derivative Prices from Webpages using a Software Agent*, 32 p., # 4, May 20, 2003.

Michael H. Breitner and Oliver Kubertin, *WARRANT-PRO-2: A GUI-Software for Easy Evaluation, Design and Visualization of European Double-Barrier Options*, 35 p., #5, September 12, 2003.

Dorothée Bott, Gabriela Hoppe and Michael H. Breitner, *Nutzenanalyse im Rahmen der Evaluation von E-Learning Szenarien*, 14 p., #6, October 21, 2003.

Gabriela Hoppe and Michael H. Breitner, *Sustainable Business Models for E-Learning*, 20 p., #7, January 5, 2004.

Heiko Genath, Tobias Brüggemann and Michael H. Breitner, *Preisvergleichsdienste im internationalen Vergleich*, 40 p., #8, June 21, 2004.

Dennis Bode and Michael H. Breitner, *Neues digitales BOS-Netz für Deutschland: Analyse der Probleme und mögliche Betriebskonzepte*, 21 p., #9, July 5, 2004.

Caroline Neufert and Michael H. Breitner, *Mit Zertifizierungen in eine sicherere Informationsgesellschaft*, 19 p., #10, July 5, 2004.

Marcel Heese, Günter Wohlers and Michael H. Breitner, *Privacy Protection against RFID Spying: Challenges and Countermeasures*, 21 p., #11, July 5, 2004.

Liina Stotz, Gabriela Hoppe and Michael H. Breitner, *Interaktives Mobile(M)-Learning auf kleinen Endgeräten wie PDAs und Smartphones*, 28 p., #12, August 18, 2004.

Frank Köller and Michael H. Breitner, *Optimierung von Warteschlangensystemen in Call Centern auf Basis von Kennzahlenapproximationen*, 24 p., #13, January 10, 2005.

Phillip Maske, Patrick Bartels and Michael H. Breitner, *Interactive M(obile)-Learning with UbiLearn 0.2*, 21 p., #14, April 20, 2005.

Robert Pomes and Michael H. Breitner, *Interactive Strategic Management of Information Security in State-run Organizations*, 7 p., #15, May 5, 2005.

Simon König, Frank Köller and Michael H. Breitner, *FAUN 1.1 User Manual*, 134 p., #16, August 4, 2005.