

Autoregressive Neural Network Processes

**Univariate, Multivariate and Cointegrated Models with
Application to the German Automobile Industry**

Inaugural-Dissertation zur Erlangung des
akademischen Grades eines Doktors
der Wirtschaftswissenschaften
der Universität Passau

von

Dipl.-Kfm. Sebastian Dietz

Oktober 2010

Outline

| | | |
|----------|--|-----------|
| 1 | Introduction | 1 |
| 1.1 | Basic Ideas and Motivation | 2 |
| 1.2 | Outlook of the Contents | 4 |
| 2 | Basic Theory of Autoregressive Neural Network Processes (AR-NN) | 6 |
| 2.1 | Time Series and Nonlinear Modelling | 6 |
| 2.1.1 | Autoregressive Processes | 6 |
| 2.1.2 | Nonlinear Autoregressive Processes | 9 |
| 2.2 | The Architecture of AR-NN | 10 |
| 2.2.1 | AR-NN Graphs | 10 |
| 2.2.2 | The AR-NN Equation | 15 |
| 2.2.3 | The Universal Approximation Theorem | 16 |
| 2.2.4 | The Activation Function | 19 |
| 2.3 | Stationarity of AR-NN | 27 |
| 2.3.1 | Stationarity and Memory | 27 |
| 2.3.2 | Markov Chain Representation and the Invariance Measure | 29 |
| 2.3.3 | Unit Roots and Stationarity of AR-NN | 31 |
| 2.3.4 | The Rank Augmented Dickey-Fuller Test | 33 |
| 3 | Modelling Univariate AR-NN | 36 |
| 3.1 | The Nonlinearity Test | 38 |
| 3.1.1 | Taylor Expansion | 38 |
| 3.1.2 | The Lagrange-Multiplier Tests | 41 |
| 3.1.2.1 | The Test of White | 42 |
| 3.1.2.2 | The Test of Teräsvirta, Lin and Granger | 45 |
| 3.2 | Variable Selection | 46 |
| 3.2.1 | The Autocorrelation Coefficient | 48 |
| 3.2.2 | The Mutual Information | 49 |
| 3.2.3 | Polynomial Approximation Based Lag Selection | 52 |
| 3.2.4 | The Nonlinear Final Prediction Error | 54 |

| | | |
|----------|---|------------|
| 3.3 | Parameter Estimation | 57 |
| 3.3.1 | The Performance Function | 58 |
| 3.3.2 | Important Matrix Terms | 61 |
| 3.3.3 | Basic Features of the Algorithms | 63 |
| 3.3.4 | First Order Gradient Descent Methods | 66 |
| 3.3.5 | Second Order Gradient Descent Methods | 70 |
| 3.3.6 | The Levenberg-Marquardt Algorithm | 71 |
| 3.3.7 | Stopped Training | 75 |
| 3.4 | Parameter Tests | 78 |
| 3.4.1 | Bottom-Up Parameter Tests | 79 |
| 3.4.1.1 | The Test of Lee, White and Granger | 79 |
| 3.4.1.2 | Cross Validation | 80 |
| 3.4.2 | Top-Down Parameter Tests | 81 |
| 3.4.2.1 | Consistency | 82 |
| 3.4.2.2 | The Neural Network Information Criterion | 86 |
| 3.4.2.3 | The Wald Test | 87 |
| 4 | Multivariate models | 88 |
| 4.1 | Multivariate AR-NN | 88 |
| 4.1.1 | Vector Autoregressive Neural Network Equations | 88 |
| 4.1.2 | Vector Autoregressive Neural Network Graphs | 91 |
| 4.2 | Neural Networks and Cointegration | 95 |
| 4.2.1 | Nonlinear Adjustment in Error Correction Models | 95 |
| 4.2.1.1 | Theoretical Prerequisites | 96 |
| 4.2.1.2 | The Nonlinear Error Correction Model and Neural Networks | 98 |
| 4.2.2 | NN-VEC graphs | 101 |
| 4.2.3 | Identifying and Testing the NN-VEC | 103 |
| 5 | The German Automobile Industry and the US Market | 105 |
| 5.1 | Economic Motivation | 106 |
| 5.2 | The Data | 109 |
| 5.3 | Nonlinearity and Stationarity Tests | 112 |
| 5.4 | Univariate AR-NN | 115 |
| 5.4.1 | Lag Selection | 115 |

| | | |
|----------|---|------------|
| 5.4.2 | Estimation and Bottom-Up Parameter Tests | 119 |
| 5.4.3 | Top-Down Parameter Tests | 135 |
| 5.4.4 | Residual Analysis | 136 |
| 5.5 | Cointegration and NN-VEC | 140 |
| 5.5.1 | The Cointegration Relationship | 140 |
| 5.5.2 | Estimation of the NN-VEC | 143 |
| 5.5.3 | Residual Analysis | 147 |
| 6 | Conclusion | 150 |
| A | Proof of Theorem 2.1 | 152 |
| B | R-Code | 154 |
| B.1 | Lag Partition Matrix | 154 |
| B.2 | Polynomial Approximation Based Lag Selection | 155 |
| B.3 | The MIC | 156 |
| B.4 | The Levenberg-Marquardt Algorithm for Univariate Models | 157 |
| B.5 | Residuals ES | 161 |
| B.6 | Fitted Values ES | 162 |
| B.7 | Prediction | 163 |
| B.8 | The Covariance Matrix | 164 |
| B.9 | The Lee-White-Granger Test | 166 |
| B.10 | Estimation of the NN-VEC | 168 |
| B.11 | Prediction with the NN-VEC | 172 |
| | Bibliography | 174 |
| | Index | 185 |

List of Figures

| | | |
|-----|--|-----|
| 2.1 | Linear AR(2) graph | 12 |
| 2.2 | AR-NN(2) graph - "black box" representation | 12 |
| 2.3 | AR-NN(2) with two hidden neurons | 14 |
| 2.4 | Reaction of certain activation functions on their input range | 22 |
| 2.5 | AR(1) with structural break | 25 |
| 2.6 | AR-NN(1) with $h=2$ approximates a TAR(1) | 25 |
| 2.7 | AR-NN(1) with $h=4$ approximates a TAR(1) | 26 |
| 2.8 | Prediction with the model from figure 2.7 | 26 |
| 3.1 | Flow chart AR-NN model building | 37 |
| 3.2 | Taylor polynomial approximation of the tanh | 41 |
| 3.3 | Iterative parameter estimation | 65 |
| 3.4 | Flow chart iterative parameter estimation | 65 |
| 3.5 | Flow chart Levenberg-Marquardt algorithm | 74 |
| 3.6 | Example: Overfitted AR-NN | 76 |
| 3.7 | Stopped training: Development of ES-RSS and VS-RSS during the learning algorithm | 77 |
| 4.1 | VAR(2) graph with 2 variables | 92 |
| 4.2 | Separated model of the first variable | 93 |
| 4.3 | VAR-NN(2) - "black box" representation | 93 |
| 4.4 | VAR-NN(2) graph | 94 |
| 4.5 | VAR-NN(2) - vector representation | 94 |
| 4.6 | Linear cointegration relationship (3 variables) | 101 |
| 4.7 | NN-VEC with 2 lags, 3 variables and 2 hidden neurons | 102 |
| 4.8 | Linear VEC with 2 lags, 3 variables | 102 |
| 5.1 | Relations between investigated variables | 108 |
| 5.2 | Data plot | 111 |
| 5.3 | AC and PAC | 117 |
| 5.4 | Univariate models in-sample plots | 127 |

| | | |
|------|--|-----|
| 5.5 | Univariate models out-of-sample plots | 128 |
| 5.6 | PCI: Surface plot AR-NN(4) with various h | 131 |
| 5.7 | EXC: Surface plot AR-NN(3) with various h | 132 |
| 5.8 | IND: Surface plot AR-NN(2) with various h | 133 |
| 5.9 | SAL: Surface plot AR-NN(2) with various h | 134 |
| 5.10 | Histogram residuals | 138 |
| 5.11 | Univariate models: Autocorrelation residuals | 139 |
| 5.12 | Cointegration relationship | 142 |
| 5.13 | NN-VEC out-of-sample plots | 146 |
| 5.14 | Histogram residuals NN-VEC(3) | 147 |
| 5.15 | Auto- and cross-correlations NN-VEC(3) | 149 |

List of Tables

| | | |
|------|--|-----|
| 2.1 | Symbols for linear AR graphs | 11 |
| 2.2 | Additional symbols for AR-NN | 13 |
| 2.3 | RADF critical values (Hallman (1990) p.39) | 34 |
| 4.1 | Additional symbols for a 2 variable VAR-NN | 92 |
| 4.2 | Additional symbols for a 3 variable NN-VEC | 101 |
| 5.1 | ADF test | 113 |
| 5.2 | Teräsvirta-Lin-Granger test χ^2 - statistic (data in first differences) . . . | 113 |
| 5.3 | Teräsvirta-Lin-Granger test F - statistic (data in first differences) . . . | 114 |
| 5.4 | Teräsvirta-Lin-Granger test χ^2 - statistic without crisis data (data in first differences, first 100 values) | 114 |
| 5.5 | MIC | 116 |
| 5.6 | Polynomial approximation lag selection | 116 |
| 5.7 | NFPE | 118 |
| 5.8 | Iterations necessary for univariate models | 122 |
| 5.9 | Lee-White-Granger test for $h=1$ | 122 |
| 5.10 | PCI: AR-NN vs. other models | 123 |
| 5.11 | EXC: AR-NN vs. other models | 124 |
| 5.12 | IND: AR-NN vs. other models | 125 |
| 5.13 | SAL: AR-NN vs. other models | 126 |
| 5.14 | PCI: Parameters AR-NN(4) with $h=4$ | 129 |
| 5.15 | EXC: Parameters AR-NN(3) with $h=4$ | 129 |
| 5.16 | IND: Parameters AR-NN(2) with $h=1$ | 130 |
| 5.17 | SAL: Parameters AR-NN(2) with $h=0$ | 130 |
| 5.18 | Univariate models: NIC | 135 |
| 5.19 | Univariate models: Wald test | 135 |
| 5.20 | Univariate models: Skewness and kurtosis | 137 |
| 5.21 | Univariate models: Jarque-Bera test | 137 |
| 5.22 | Univariate models: Box-Pierce test | 137 |

| | | |
|------|---|-----|
| 5.23 | Univariate models: ARCH-LM test (χ^2 - statistic) | 138 |
| 5.24 | Cointegrated NN with varying h | 144 |
| 5.25 | Parameters NN-VEC(3) | 145 |
| 5.26 | NN-VEC(3): Skewness and kurtosis | 147 |
| 5.27 | NN-VEC(3): Jarque-Bera test | 148 |
| 5.28 | NN-VEC(3): Box-Pierce test | 148 |
| 5.29 | NN-VEC(3): ARCH-LM test (χ^2 - statistic) | 148 |

1 Introduction

Prediction of future values of economic variables is a basic component not only for economic models, but also for many business decisions. It is difficult to produce accurate predictions in times of economic crises, which cause nonlinear effects in the data. In the following a new statistical method is introduced, which tries to overcome the problem of such nonlinear effects.

This dissertation belongs to the scientific field of time series analysis, an important subfield of econometrics. The aim of time series analysis is to extract information of a given data series, consisting of observations over time. This information is used to build a model of the dynamics, called process, which determines the data series. Such a model can be used for prediction of future values of the time series. For identification of the process linear models like linear autoregressive processes (AR) and autoregressive moving average processes (ARMA) are a standard tool of econometrics at least since Box and Jenkins (1976). In particular Wold's theorem (Wold (1954)) has popularized ARMA. However empirical experience shows that linear models are not always the best way to identify a process and do not always deliver the best prediction results. In this context Granger and Teräsvirta (1993) speak of "hidden nonlinearity", which requires the adoption of nonlinear methods. Particularly in times of economic crises nonlinearities may appear. Since the early 1990's a lot of nonlinear methods have arisen. They can be divided into parametric models, characterized by a fixed number of parameters in a known functional form, and the more general nonparametric models.

The method for nonlinear time series analysis discussed in this dissertation - autoregressive neural network processes (AR-NN) - is parametric. Due to this, it has all the advantages concerning estimation and testing connected with parametric methods. In addition AR-NN fulfill the requirements for the universal approximation theorem of neural networks in Hornik (1993). Thus they are able to approximate any unknown nonlinear process. A bottom-up strategy for model building makes them applicable to typical economic time series. Hence the prediction of economic time series can be improved with AR-NN. The theory is not constrained to univariate time series models

only, but can also be extended to multivariate and vector error correction models.

The contribution of this dissertation to science is the discussion of a nonlinear method for analysis of nonlinear economic time series, which is able to produce better results in out-of-sample prediction, because of its universal approximation property of neural networks. The method is parametric and can be handled like the well known linear methods in time series analysis: The models can be built according to the steps of Box and Jenkins (1976) (data preparation, variable selection, parameter estimation and parameter tests) using some nonlinear methods, proposed in this dissertation, for each step. The following section shortly introduces the basic ideas and the motivation and section 1.2 gives a summary of the contents.

1.1 Basic Ideas and Motivation

Here a method for the analysis of economic time series is introduced, which is based on artificial neural networks, a class of functions which became popular in many fields of science from the late 1980's to the late 1990's. Certainly statistics is not the only application area for neural networks. But used as statistical function they seem particularly interesting, for diverse authors have shown that certain artificial neural networks can approximate any function (universal approximation theorem, see Cybenko (1989), Funahashi (1989), Hornik, Stinchcombe and White (1989), Hornik (1991), Hornik (1993), Liao, Fang and Nuttle (2003)). Various artificial neural networks have been used for analysis of economic or financial time series (examples are White (1988), White (1989b), Gencay (1994), Kuan and White (1994), Kaastra and Boyd (1996), Swanson and White (1997), Anders, Korn and Schmitt (1998), Medeiros, Teräsvirta and Rech (2006) to mention just a few). In contrast to the mentioned works, which sometimes include high parametrized and complicated models, we want to improve linear AR's with elements of neural networks using a bottom-up strategy. The starting point is basically a linear AR. Only if a nonlinearity test indicates hidden nonlinearity, nonlinear components are added. Further more also the complexity of the nonlinear part of the models is increased step- by- step, always using tests which indicate if additional elements might contribute significantly to the performance of the models. Thus we call our AR-NN here **augmented**. The aim of such a procedure is to keep the models as simple as possible. As a consequence AR-NN are not only applicable to high-frequency data usually

connected with neural networks, but also to time series with around 100 observations, which are typical in economics.

Additionally, we use three other properties of neural networks which are sufficient for the universal approximation property: The networks are only **feedforward** directed, consist of only **three layers** and the nonlinear part is based on a **bounded** nonlinear function. The first two properties keep the structure of the processes straightforward, the third property contributes to analyze the stability behavior of the process (stationarity). The consequence are simple structured processes, consisting of a linear and a nonlinear part, which are adaptable for the “classical” steps for modelling time series (model selection, parameter estimation and parameter tests, see Box and Jenkins (1976) part II). On the other hand our models have all the advantages of neural networks. The most important of them is the ability to handle any nonlinearity. In the empirical part (chapter 5) it is shown that AR-NN sometimes perform better than some popular linear as well as nonlinear alternatives concerning the out-of-sample performance.

So far the existing literature has already been discussing particular problems and can be combined for modelling procedures of AR-NN, but until now multivariate and multivariate cointegrated processes have not been modeled using artificial neural networks. We introduce such multivariate modelling and show how the nonlinear vector- error- correction model of Escribano and Mira (2002) can be concretized using neural networks. The result is linear cointegration with nonlinear adjustment. Such neural network error correction is necessary if the time series involved in a cointegration relationship are nonlinear. In such a model a linear cointegration relationship between some nonlinear variables is adjusted at the variables via nonlinear error- correction. An example for application are supply- demand equations: Let the supply as well as the demand data be a nonlinear time series, whereas the equilibrium between supply and demand is linear. Let the long-run equilibrium between the series be a cointegration relationship. Now for prediction of the individual series using a vector error correction model, the long-run linear cointegration relationship has to be adjusted at the nonlinear series, because the long-run equilibrium has an individual nonlinear influence on each series. The results are better predictions than with linear error correction models.

To put it in a nutshell, AR-NN as proposed in this dissertation are processes which combine “classical” time series analysis with the advantages of artificial neural networks,

taking into account to keep the models as simple as possible. Those processes are able to handle hidden nonlinearity, which appears in economic time series, in particular, in times of economic crises and changes. In contrast to many other works neural networks here are not considered to be a black box, but rather a parametric statistical function which is able to include nonlinear phenomena. In this context AR-NN are improved linear models rather than pure nonlinear models.

1.2 Outlook of the Contents

The structure of chapters 2 and 3 follows the steps necessary for adjusting a univariate model at a time series. Chapter 2 introduces univariate AR-NN and explains their properties. Therefore, in section 2.1 at first the basic theory of time series analysis is introduced and connected to the ideas of nonlinear modelling. The subsequent sections show how a linear model is extended for nonlinear components (so called hidden neurons) to receive an AR-NN. The components of the AR-NN equation (linear and nonlinear part) are explained in graph form as well as a written description. An interesting and important point is the stability behavior of AR-NN. Using the results of Trapletti, Leisch and Hornik (2000) we show that only the linear part determines the stationarity of AR-NN. Therefore a modification of the well known Augmented Dickey-Fuller test, the Rank Augmented Dickey-Fuller test, can be applied as a stationarity test.

Chapter 3 provides the tools necessary for model selection, parameter estimation and parameter tests. Only the finding of nonlinearity in a given time series justifies the use of nonlinear methods. Hence nonlinearity tests have to be applied before the nonlinear model is adjusted. The first part (section 3.1) introduces the nonlinearity tests. Section 3.2 shows four methods of selecting the lag order for nonlinear models. In the subsequent sections the numerical parameter estimation methods usually used for neural networks are introduced. In particular the Levenberg-Marquardt algorithm seems to be the best solution: It combines the advantages of first and second order gradient descent methods. Section 3.4 explains how parameters of the nonlinear model can be tested for significance.

Chapter 4 indicates how the theory from chapters 2 and 3 can be transferred to multivariate and cointegrated models. For cointegrated models the nonlinear error correction

theorem of Escribano and Mira (2002) is used. It can be interpreted as linear cointegration with nonlinear adjustment. As for the univariate models, the Levenberg-Marquardt algorithm can also be used for parameter estimation. Graphical representation is used to explain the complicated connections between the components of the multivariate and cointegrated models.

In chapter 5 the theory is applied to real economic data. Four variables connected with the German automobile industry are used: The industrial production of car manufacturers in Germany, the sale of imported foreign automobiles in the USA, the Dollar to Euro exchange rate and an index of selected German car manufacturers stocks. Data are provided on a monthly basis from January 1999 to October 2009. The number of observations (129) is typical for economic time series. Although neural networks are usually used for larger datasets, we show that a bottom-up arranged AR-NN - starting with a linear process - may deliver quite good results for the given short time series. In the first part of this chapter a univariate nonlinear model is adjusted to each series. The out-of-sample performance is measured at a subset of the data set which includes obvious nonlinearities caused by the economic crisis since the end of 2008. A one- and an eight- period forecast is compared to some other linear as well as nonlinear methods. In section 5.5 a nonlinear error correction model using neural networks is estimated. Univariate AR-NNs as well as the error correction model perform quite well compared to some linear as well as nonlinear alternatives concerning the out-of-sample performance.

Nearly all theory used in the empirical part has been implemented in the statistical programming language R. The programming code is provided in appendix B. Concerning this code one remark has to be made: Keeping the functions general was not always possible. Therefore some of the functions can only be used with the data set used in this dissertation or at least similar data sets.

2 Basic Theory of Autoregressive Neural Network Processes (AR-NN)

2.1 Time Series and Nonlinear Modelling

This section introduces the basic theory of autoregressive processes (AR). We start with a definition of AR. In contrast to most of the other time series literature we use a general definition to ensure that nonlinear autoregressive processes are also autoregressive processes by their basic properties. Furthermore an introduction is given to the problems in linear estimation and the aims of nonlinear models to overcome them. We distinguish between parametric, semiparametric and nonparametric nonlinear methods.

Most nonlinear models are dedicated to certain specific nonlinearities in the data (like structural breaks in the regression coefficient or constant), while AR-NN are able to approximate any function and therefore any nonlinearity (see section 2.2.3). As it is shown in this section, they are parametric, which makes them easy to handle. These two features are the main reasons why neural networks are used in this dissertation to overcome the problem of hidden nonlinearity.

2.1.1 Autoregressive Processes

Time series analysis, a subfield of econometrics, is engaged in analyzing the underlying dynamics of a set of successively observed past time values, called time series. We call the underlying dynamics stochastic process and describe it as a series of random variables $\{\tilde{x}_t\}_{t=1}^T$ with finite time index $t = 1, 2, \dots, T$. A time series is a series $\{x_t\}_{t=1}^T$ of observed realizations of the random variable (see for example Wei (1990) pp. 6-7). Actually only the time series is given. We want to identify the process which determines the time series using only the information given by the series. Therefore the process is separated into a part which we can determine or predict and a random part. To create a useful model of a process, as much as possible should be explained by the first part

and the latter should be kept as small as possible. Usually the first part determines the expectation conditioned by certain exogenous variables and the random part is accountable for the deviations, or in other words the variance. Thus a variance minimal model means that the predictable part explains as much as possible of the time series.

The simplest and probably the most common way is to construct the process as a function of n past observed values of the time series. Because this implies that one usually estimates this function by regressing x_t on its past values, such a process is called AR. Formally it is introduced in definition 2.1.

Definition 2.1 (Autoregressive process):

A process is called autoregressive process of order n , short $AR(n)$, if it is represented by the equation

$$x_t = F(X_{t-1}) + \varepsilon_t, \quad (2.1.1)$$

whereas $X_{t-1} = (x_{t-1}, x_{t-2}, \dots, x_{t-n})^\top$, $F : \mathbb{R}^n \rightarrow \mathbb{R}$ and ε_t is a i.i.d. $N(0, \sigma^2)$ (Gaussian WN) random variable. The first term on the right hand side of equation (2.1.1) is called predictable part, the second term stochastic part.

Remark 2.1.1:

If $F(X_{t-1})$ is a linear function, the process is a linear AR. If $F(X_{t-1})$ is nonlinear it is a nonlinear AR.

The influence of the stochastic part is only of temporary nature and contains no time dependent trends or a variance (heteroskedasticity) as σ^2 is finite and equal $\forall t$. Note that in definition 2.1 ε_t is simply added to the predictable part. Of course in theory it is also possible to combine the predictable and the stochastic part multiplicatively. However, this is not very common and probably not feasible. Thus we exclude multiplicative errors in definition 2.1. It also has to be mentioned that we only deal with the constant distance one between the lags.

The conditional expectation of x_t is defined as $E(x_t|X_{t-1}) = F(X_{t-1})$ and the conditional expectation of ε_t is defined as $E(\varepsilon_t|X_{t-1}) = 0$. This means that the input and

the stochastic part ε_t are completely uncorrelated. If a process is an $AR(n)$ we can say that the process has a memory in mean which goes back until period n . It is important to know that any specification of the predictable part requires stationarity of the time series as spurious regression could occur otherwise (Granger and Newbold (1974)). For definition, testing and preprocessing concerning stationarity see section 2.3.

Linear AR are the most simple and oldest models for processes, first mentioned in Yule (1927). In full representation a linear $AR(n)$ is written as

$$x_t = \alpha_0 + \alpha_1 x_{t-1} + \alpha_2 x_{t-2} + \dots + \alpha_n x_{t-n} + \varepsilon_t \quad (2.1.2)$$

Application shows that in most cases the residuals hardly match the Gaussian WN assumption. A linear solution of this problem are the ARMA processes, see Box and Jenkins (1976) p.11. They assume that the process does not only consist of a linear predictable part and an additive Gaussian WN. Rather the stochastic part itself may be determined by a moving average process (MA) of the Gaussian WN ε_t . An $ARMA(n, k)$ process is represented by the following equation (k indicates the maximum lag of the MA part):

$$x_t = \alpha_0 + \alpha_1 x_{t-1} + \alpha_2 x_{t-2} + \dots + \alpha_n x_{t-n} + \varepsilon_t + \alpha_1^* \varepsilon_{t-1} + \dots + \alpha_k^* \varepsilon_{t-k} \quad (2.1.3)$$

Until today ARMA are the most frequently applied process models in time series analysis. The Wold decomposition theorem (introduced in Wold (1954)) justifies theoretically that one can estimate any covariance stationary process by an ARMA. However according to Lütkepohl and Tschernig (1996) p.149. ARMA are only the best linear estimators. In practical application however sometimes even large ARMA are inferior to simple linear AR concerning the out-of-sample performance, because they are not able to capture nonlinearities like regime effects and tend to overfitting. Sometimes logarithms may help to linearize some nonlinear effects, but information can be lost by the transformation. A linear solution might be to extend the assumptions on the stochastic part, particular the Gaussian distribution. The alternative are nonlinear models (see Fan and Yao (2003) p.15).

2.1.2 Nonlinear Autoregressive Processes

Nonlinear models try to overcome the problem of observed nonstandard features¹ in linear models. They can be interpreted as an alternative draft to linear models with extensions on the stochastic part (ARMA) as they try to improve the predictable part to explain the process rather than to add some stochastic components or to introduce some assumptions which are difficult to handle. By contrast it is possible that a nonlinear AR has its ε_t in accordance with the standard assumptions in definition 2.1. In natural sciences only nonlinear modelling allows us to think of pure deterministic processes (which for example chaos science tries to analyze). However according to Granger and Teräsvirta (1993) p.2 such theory does not fit to economic and financial time series. Nonlinear methods are more flexible than linear models on the one hand, but it may become difficult to interpret their parameters (Medeiros, Teräsvirta and Rech (2006) p.49).

The entirety of nonlinear modelling techniques is large. The first step to classify them is to distinguish between parametric, semiparametric and nonparametric methods. Parametric means that the structure of the function to estimate and the number of the related parameters are known. Examples are threshold autoregression (TAR) or smooth transition autoregression (STAR), methods which consider regime switching effects. Nonparametric models do not constrain the function to any specific form, but allow for a range of possible functions. Kernel regression for example would belong to this class. Granger and Teräsvirta (1993) p.104 describe semiparametric models as a combination of parametric and nonparametric parts. Granger and Teräsvirta (1993) p.105 as well as Kuan and White (1994) p.2 classify neural networks as parametric econometric models, for the model has to be specified - including the number of parameters - before it is estimated.

As we will see below neural networks have a universal approximation property. This means that they are able to approximate any (not specified) function arbitrary accurately. This property can be seen as evidence for a nonparametric model. However, the neural network function has to be specified and is therefore parametric, even if this parametric function may be able to approximate any unknown function arbitrary

¹The term nonstandard features means the same as hidden nonlinearity and is used by Fan and Yao (2003) p.15

precisely. Hence a neural network can be referred as parametric model in the statistical sense (see Anders (1997) p.185). Of course in estimating linear functions neural networks are clearly inferior to linear methods because of the needless additional effort.

2.2 The Architecture of AR-NN

Neural networks as we will use them and as they appear often in econometric literature always contain a linear and a nonlinear part. To make the neural network function easily accessible, we use signal-flow graph representation, stepwise, at first of the linear part and then of the whole neural network function. For the usage in the subsequent chapters we introduce vector representation of the scalar neural network function. We explain the basic components of the universal approximation theorem in the version of Hornik (1993). As the universal approximation property depends of the activation function, we discuss some appropriate bounded functions. Their boundedness allows the analysis of stationarity using linear methods as we will see in section 2.3. Non-bounded activation functions in contrast are much more difficult to handle. After the activation function and the architecture of the network including the number of parameters is specified, the AR-NN becomes a parametric function as mentioned above. This is the starting point for model building according to the typical scheme of Box and Jenkins (1976) part II (variable selection, estimation, evaluation) in the subsequent chapter.

2.2.1 AR-NN Graphs

Graphical visualization is the first step to understand the AR-NN function. The graphs we use here are architectural graphs, similar to those in Anders (1997) or Haykin (2009) for example. They serve as "blueprint" of the models and give some deeper insight into complicated networks.² This will be particularly useful if the models become more complex (see chapter 4). At first we start with the graph of a linear AR. The elements we need and their equivalents in functional representation are shown in table 2.1.

In linear time series analysis the term layer is unknown. For the graph of the linear AR we need two layers: The input layer, which contains the entirety of all independent variables and the output layer, which contains the dependent variables (only one variable

²For design of the graphs the software yEd Graph Editor was used.

in the univariate case). Note also that the constant term is decomposed into a bias neuron with value 1 and the bias parameter α_0 . This serves for easier representation, in particular if the models contain more than one constant in the following. A graph of a linear AR(2) given by

$$\widehat{x}_t = \alpha_0 + \alpha_1 x_{t-1} + \alpha_2 x_{t-2} \quad (2.2.1)$$

is shown in figure 2.1. We abstain here from the stochastic part as we only deal with an estimator \widehat{x}_t which corresponds to the (conditional) expectation of the process x_t (the expectation of the stochastic part is 0).





| Symbol | Statistical term | Term in NN theory | Equivalent in functions |
|---|------------------|--------------------------|-------------------------------------|
|  | Variables | Input and output neurons | \widehat{x}_t, x_{t-i} |
|  | Parameters | Shortcut weights | α_i |
|  | Constant | Bias | 1 (to be multiplied by α_0) |
|  | - | Layer | - |

Table 2.1: Symbols for linear AR graphs

As we know from the introduction, a linear AR is sometimes not sufficient and has to be augmented therefore for a nonlinear part. The entirety of this nonlinear part is called the “hidden” layer. It is inserted between the input- and output layer. This basic concept is shown in figure 2.2: Inside the nonlinear layer the variables are transformed by a nonlinear function. The result of this nonlinear transformation is added to the result of the linear part. Let $F(\cdot)$ be such a nonlinear function (it will be concretized later), then the nonlinear extension of an linear AR(2) (as in equation (2.2.1)) is given by

$$\widehat{x}_t = \alpha_0 + \alpha_1 x_{t-1} + \alpha_2 x_{t-2} + F(x_{t-1}, x_{t-2}). \quad (2.2.2)$$

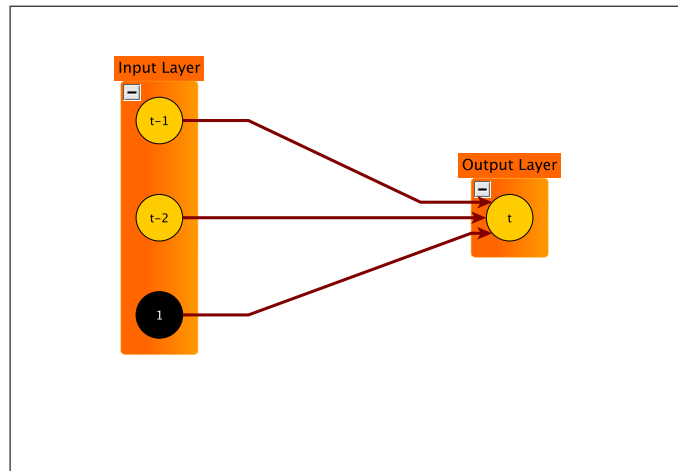


Figure 2.1: Linear AR(2) graph

Source: Authors' design

The nonlinear part here is described as a "black box", which generates some contribution to the result, but is not yet known. Figure 2.2 shows the graph belonging to the equation (2.2.2), whereas $F(x_{t-1}, x_{t-2})$ is represented by the hidden layer.

Now we will have a look inside the hidden layer. To understand the nonlinear trans-

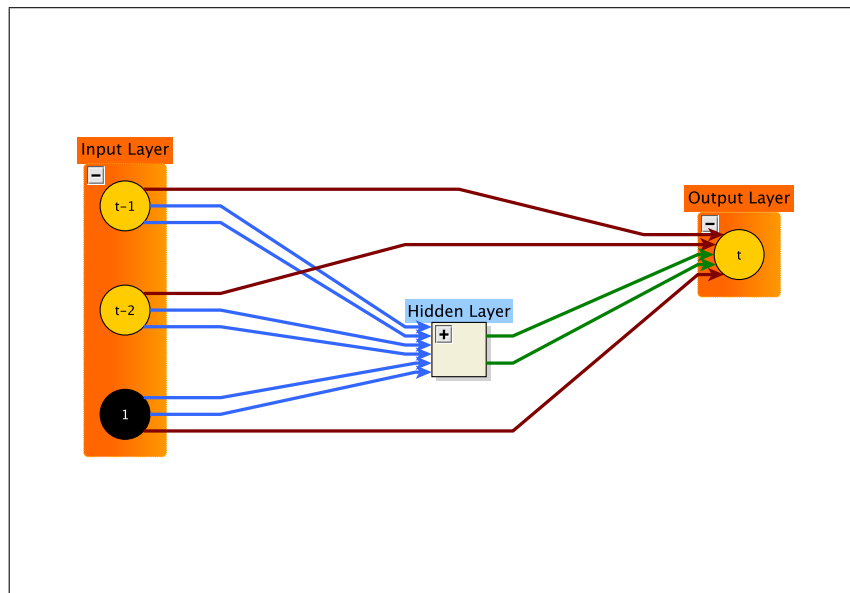


Figure 2.2: AR-NN(2) graph - "black box" representation

Source: Authors' design

formation, a few additional symbols are necessary. They are defined in table 2.2. The nonlinear part contains h so called hidden neurons, which transform the input variables, weighted by parameters γ_{ij} plus a bias γ_{0j} , via a nonlinear activation function $\Psi(\cdot)$. Let

i indicate the number of lags and j the number of hidden neurons. A hidden neuron is denoted by

$$\Psi \left(\gamma_{0j} + \sum_{i=1}^n \gamma_{ij} x_{t-i} \right). \quad (2.2.3)$$

Each hidden neuron is weighted by a parameter β_j before it belongs to the output layer. Assume that $h = 2$, then the nonlinear part $F(x_{t-1}, x_{t-2})$ in equation (2.2.2) becomes

$$F(x_{t-1}, x_{t-2}) = \Psi(\gamma_{01} + \gamma_{11}x_{t-1} + \gamma_{21}x_{t-2})\beta_1 + \Psi(\gamma_{02} + \gamma_{12}x_{t-1} + \gamma_{22}x_{t-2})\beta_2. \quad (2.2.4)$$

In the most cases $\Psi(\cdot)$ is the same for all hidden neurons, but it also can be chosen to be different for each hidden neuron. However this is not common practise and leads to complications in the estimation procedures. Now we can unveil the “black box” in our graph (see figure 2.3) and substitute $F(x_{t-1}, x_{t-2})$ in equation (2.2.2) by equation (2.2.4).




| Symbol | Description | Equivalent in functions |
|---|--|--|
|  | Weight between input- and hidden neuron | γ_{ij} |
|  | Weight between hidden- and output neuron | β_j |
|  | Hidden neuron: Returns a nonlinear transformation of the weighted input neurons | $\Psi \left(\gamma_{0j} + \sum_{i=1}^n \gamma_{ij} x_{t-i} \right)$ |

Table 2.2: Additional symbols for AR-NN

In the further procedure all AR-NN are constructed like the one in figure 2.3: Forward directed (all edges are forward directed in the graphs) with only one hidden layer. Those properties are sufficient to guarantee the universal approximation property of the

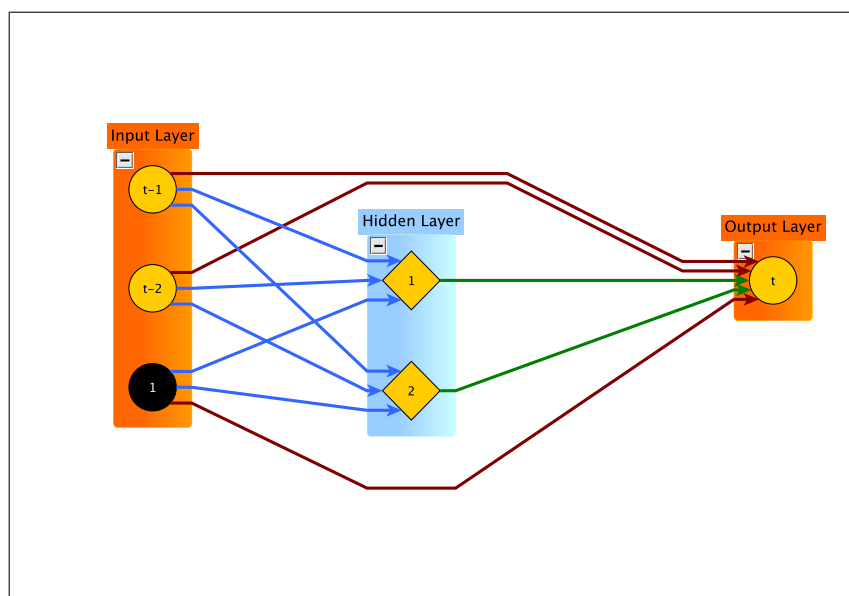


Figure 2.3: AR-NN(2) with two hidden neurons

Source: Authors' design

networks (see section 2.2.3 for details). In particular for our time series models multi hidden layer or recursive neural networks will probably add not much additional value, as such neural networks become very complicated, with impact on parameter estimation etc.

We will see below that single hidden layer feedforward networks are sufficiently to estimate any function - if the number of hidden neurons is sufficient large. Hence, also more complicated neural networks (like multi hidden layer) functions can be approximated by a single hidden layer neural network. Empirical application in chapter 5 shows that for some series with around 130 observations models with 1-4 hidden neurons improve the out-of-sample performance compared to some alternative linear models. More hidden neurons probably do not contribute any additional value. Thus multilayer neural networks with large numbers of parameters might be just "too much" for economic data series and lead to overfitted and therefore senseless models.

2.2.2 The AR-NN Equation

Once knowing the structure of AR-NN from graphs, it is easy to formulate the scalar AR-NN equation. For the AR-NN(2) the full network representation is

$$\begin{aligned}\widehat{x}_t &= \alpha_0 + \alpha_1 x_{t-1} + \alpha_2 x_{t-2} + \\ &\quad \Psi(\gamma_{01} + \gamma_{11} x_{t-1} + \gamma_{21} x_{t-1}) \beta_1 + \\ &\quad \Psi(\gamma_{02} + \gamma_{12} x_{t-1} + \gamma_{22} x_{t-1}) \beta_2.\end{aligned}\quad (2.2.5)$$

If the stochastic part is included we can write

$$x_t = \alpha_0 + \sum_{i=1}^n \alpha_i x_{t-i} + \sum_{j=1}^h \Psi \left(\gamma_{0j} + \sum_{i=1}^n \gamma_{ij} x_{t-i} \right) \beta_j + \varepsilon_t. \quad (2.2.6)$$

In the literature (for example Granger and Teräsvirta (1993) p.125) sometimes neural networks without a linear part can be found. A linear part (also called shortcut connections) is always included here, as our philosophy is - as already mentioned in the introduction - to improve linear models by augmenting them for a nonlinear part if a nonlinearity test shows that there is hidden nonlinearity in the data.

Particularly for estimation, it makes sense to write equation (2.2.6) in vector representation with vector input and scalar output. Therefore the following notations are introduced:

$$A = (\alpha_1, \alpha_2, \dots, \alpha_n)^\top$$

$$\Gamma_j = (\gamma_{1j}, \gamma_{2j}, \dots, \gamma_{nj})^\top$$

$$\Theta = (\alpha_0, A^\top, \gamma_{01}, \Gamma_1^\top, \beta_1, \dots, \gamma_{0h}, \Gamma_h^\top, \beta_h)^\top$$

The dimension of Θ is $(r \times 1)$ with $r = (n + 2) \cdot h + n + 1$. The first version of the vector representation of equation (2.2.6) is

$$x_t = \alpha_0 + A^\top X_{t-1} + \sum_{j=1}^h \Psi(\gamma_{0j} + \Gamma_j^\top X_{t-1}) \beta_j + \varepsilon_t. \quad (2.2.7)$$

Using Θ the short representation of the AR-NN equation (2.2.6) is

$$x_t = G(\Theta, X_{t-1}) + \varepsilon_t. \quad (2.2.8)$$

Finally some considerations concerning the selection of the number of the hidden neurons: A usual approach is to specify the network for an arbitrary number of hidden neurons and later test the significance of each hidden neuron (see the testing procedures in section 3.4). A common rule of thumb is to set the number of hidden neurons equal to the median of input and output variables (here: $h = (n + 1)/2$), see Anders (1997) p.104. Of course this method does not account for any technical needs like data specific behavior or the reaction of the activation function on the inputs. Hence it is not really a practical tool. White (1992) says that the number of observations of the input variables should not exceed the number of parameters by the factor 10 ($r = T/10$) to avoid overparametrization.

A method consistent with the procedure to augment a linear AR for a nonlinear part - if the data are nonlinear - is to extend the number of hidden neurons step by step: At first only one hidden neuron is added, then it is tested by a bottom-up parameter test (see section 3.4.1) to see if an additional hidden neuron would improve the model. If the test gives evidence for this, the additional hidden neuron is added. This procedure can be repeated several times until a model with a sufficient number of hidden neurons is reached.

2.2.3 The Universal Approximation Theorem

The universal approximation property was independently detected at first only for certain activation functions by Cybenko (1989), Funahashi (1989) and Hornik, Stinchcombe and White (1989). Hornik (1991) proved that any continuous, bounded and nonconstant activation function can approximate any function on a compact set \mathbf{X} (see below) if sufficient hidden units are implemented (with respect to a certain distance measure). Finally Hornik (1993) weakened the conditions for the activation functions, which should at least be locally Riemann integrable and nonpolynomial. This means that the universal approximation property of neural networks does not depend on any specific activation function, but rather on the network structure (Hornik (1991) p.252). In the following we analyze the formulation of the universal approximation theorem according to

Hornik (1993).³ Note that the universal approximation theorem does not depend of any linear components. Its focus is only on the approximation of the nonlinear part or hidden nonlinearity, which is not covered by the linear function (and therefore represented by function $F(\cdot)$ in equation (2.2.2)).

First some notations have to be introduced: Let $\mathcal{W} \subseteq \mathbb{R}^n$ be the weight space such that all $\Gamma_j \in \mathcal{W}$ and $\mathcal{B} \subseteq \mathbb{R}$ the bias space such that all $\gamma_{0j} \in \mathcal{B}$. Then $\mathcal{G}(\Psi; \mathcal{B}, \mathcal{W})$ is the set of all functions of the form

$$G(\Theta, X_{t-1}) = \sum_{j=1}^h \Psi(\gamma_{0j} + \Gamma_j^\top X_{t-1}) \beta_j, \quad (2.2.9)$$

which estimate the "true" $F(X_{t-1})$. In other words, $\mathcal{G}(\Psi; \mathcal{B}, \mathcal{W})$ is the set of all functions which can be implemented by a neural network with biases in \mathcal{B} and first to second layer weights in \mathcal{W} . Let \mathbf{X} be the n -dimensional input set. Let $\mathcal{F}(\mathbf{X})$ denote the space of all continuous functions with fix n , $F(X_{t-1})$, on the input set. Further we need the term nondegenerate: An interval is said to be nondegenerate if it has positive length. The performance or density of an estimation function is measured with respect to the input environment measure $\mu(\mathbb{R}^n) < \infty$ and some p , $1 \leq p < \infty$ by the distance

$$\rho_{p,\mu}(F, G) = \left(\int_{\mathbb{R}^n} |F(X_{t-1}) - G(\Theta, X_{t-1})|^p d\mu(X_{t-1}) \right)^{\frac{1}{p}}. \quad (2.2.10)$$

Usually one chooses $p = 2$, therewith equation (2.2.10) is equal to the mean-squared error (see Hornik (1991) p.251). We call the subset \mathcal{G} of $\mathcal{F}(\mathbf{X})$ **dense in $\mathcal{F}(\mathbf{X})$** if $\rho_{p,\mu}(F, G) < \epsilon$ with an arbitrary function $G \in \mathcal{G}$ and a number $\epsilon > 0$. Therewith we can formulate the universal approximation property by

Theorem 2.1 (Hornik (1993) p.1069 theorem 1):

Let $\Psi(\cdot)$ be Riemann integrable and nonpolynomial on some nondegenerate compact interval \mathcal{B} and let \mathcal{W} contain a neighborhood of the origin. Then $\mathcal{G}(\Psi; \mathcal{B}, \mathcal{W})$ is dense in $\mathcal{F}(\mathbf{X})$.

PROOF: See appendix A for a sketch of the proof. For the original proof see Hornik (1993)

³We concentrate on that version of the universal approximation theorem (Hornik (1993)) because it probably covers the widest range of activation functions.

pp.1070-1072.

Remark 2.1.2:

In Hornik (1993) p.1069 instead of the formulation " $\mathcal{G}(\Psi; \mathcal{B}, \mathcal{W})$ is dense in $\mathcal{F}(\mathbf{X})$ " an expression using topological terms is used: " $\mathcal{G}(\Psi; \mathcal{B}, \mathcal{W})$ contains a subset that contains $\mathcal{F}(\mathbf{X})$ in its closure with respect to uniform topology". Both mean the same, see White (1992) p.21.

The term nonpolynomial is needed because only polynomials up to a certain degree can be implemented in finite layer networks (Hornik (1993) p.1070). The universal approximation property is implied in theorem 2.1 by the fact that by any function $G(\Theta, X_{t-1})$ one can approximate any $F(X_{t-1})$ up to a certain finite number ϵ , provided that some conditions are met. Thus the aim of modelling AR-NN is to approximate this function as best as possible, which means trying to minimize ϵ as much as possible. On the one hand a large number of h might lead to that goal. On the other hand algorithms which choose the parameter vector Θ in an intelligent way are necessary to minimize ϵ . The universal approximation theorem itself says nothing about the existence of a unique solution of the approximation problem or about the estimation procedures for the neural network (Widmann (2000) p.21).

Universal approximation has its limits in so far as one can only estimate but not identify any function. If the true function is linear or polynomial, the corresponding methods may behave much better than a neural network. A critical point is also the number of hidden neurons. The more hidden neurons and consequently parameters that are introduced, the more complex the neural network becomes. Therefore there is a conflict of objectives between avoiding overparametrization and precision.

So far we have seen that universal approximation is possible using an AR-NN without linear part. This result is essential for the purpose in identifying the additional hidden nonlinearity in a process. Consider equation (2.2.2) with a not specified nonlinear function $F(x_{t-1}, x_{t-2})$. No matter what kind of equation it might be, the hidden neurons in the AR-NN can approximate it.

Caution has also to be paid to the number of hidden units. For example Lütkepohl

and Tschernig (1996) p.164 generate data with a linear AR(3) and estimate the process by an AR-NN with varying number of hidden neurons, $h = 0, \dots, 5$. They calculate the in-sample and out-of-sample standard deviation of the residuals for each model. If one chooses the out-of-sample performance as a decision criterion, a model with $h = 1$ is optimal (and thus the linear model is not identified). Consequently the neural network has only approximated, not identified the true equation. This fact intuitively says that the AR-NN can be a misspecified model, which is nevertheless able to give a good approximation.

2.2.4 The Activation Function

The next step is to specify the activation function $\Psi(\cdot)$. Determining the activation function is the first step to concretize and thus to parametrize the AR-NN function. In the sense of statistical model building the borderline between semiparametrism and parametrism is therewith crossed. We have seen in the subsection above, that the universal approximation property does not depend on any certain activation function. The only prerequisite is nonpolynomiality and Riemann integrability (as far as \mathbf{X} is compact of course). In later sections we will see that boundedness of the activation function is necessary for analysis of stationarity. Hence only bounded activation functions will be needed in the further proceedings. Concerning the Riemann integrability there should be no conflict with the bounded activation functions we use and theorem 2.1, as a Riemann integrable function has to be bounded and continuous or monotone respectively (see for example Carathéodory (1927) p.463). We abstain from using radial basis function (RBF) neural networks. They differ from the usual AR-NN by the different calculation of the nonlinear part. Compared to the neural networks we use, RBF-networks are more complicated to estimate as they contain an additional bandwidth parameter. As RBF-networks resemble strongly to kernel regression, intuitively they can be classified as semi- or even nonparametric functions. In addition it might be difficult to analyze stationarity if RBF are used. Hence the relationships between AR-NN with RBF activation functions and linear AR are not as big as the relationships between linear AR and AR-NN with the activation functions proposed below. For an application of RBF in analysis of financial time series see for example Hutchinson (1994).

The best known bounded activation functions are the so called sigmoid functions. They

are called sigmoid because of their "S"-like plot. The first one of the sigmoid functions is the logistic function

$$\Psi_{logistic}(\cdot) = (1 + e^{-\cdot})^{-1}, \quad (2.2.11)$$

$\Psi_{logistic} : \mathbb{R} \rightarrow [0; 1]$. Another well known sigmoid function is the tangens hyperbolicus (*tanh*)

$$\Psi_{tanh}(\cdot) = \frac{e^{\cdot} - e^{-\cdot}}{e^{\cdot} + e^{-\cdot}}, \quad (2.2.12)$$

$\Psi_{tanh} : \mathbb{R} \rightarrow [-1; 1]$. Note that the *tanh* can be calculated out of the logistic function by

$$\Psi_{tanh}(\cdot) = 2\Psi_{logistic}(2(\cdot)) - 1,$$

so it is inessential, which function is used (Widmann (2000) p.16). According to Dutta, Ganguli and Samanta (2005) p.5 sigmoid functions reduce the effect of outliers, because they compress the data at the high and low end. Such functions also can be called squashing functions (Castro, Mantas and Benitez (2000) p.561). Although in the literature often only sigmoid and RBF activation functions are considered, it is also possible to choose any other bounded, Riemann integrable and nonpolynomial activation function. The cosine is also sometimes used, for example in Hornik, Stinchcombe and White (1989). Like the sigmoid activation functions it has also a bounded range of values. Far less common are the Gaussian,

$$\Psi_G(\cdot) = e^{-\frac{1}{2}(\cdot)^2} \quad (2.2.13)$$

and the Gaussian complement activation function

$$\Psi_{GC}(\cdot) = 1 - e^{-\frac{1}{2}(\cdot)^2}, \quad (2.2.14)$$

which both map on the unit interval $[0; 1]$. The choice of the activation function may be useful if additional information on the process is available or one wants to gain a certain effect (see Dutta, Ganguli and Samanta (2005) p.5). The Gaussian and the Gaussian complement function underline the effect of values in the middle range. Nevertheless as we have seen above, the universal approximation theorem states that the universal approximation property of an AR-NN does not depend upon any specific activation function.

Because of the bounded value range of certain activation functions, scaling of the

data set on those intervals could be useful sometimes, but it is not necessary. However scaling of the data set has two main advantages (see also Anders (1997) pp.25-26):

- The learning procedures (see section 3.3) behave much better if the variables are scaled. In particular if the range of the observed values is much bigger than the range of the activation function, the linear part may dominate the whole process. As a consequence, the result is similar, or at least not much better than a linear AR. On the other hand if the data already range in an interval close to the interval at which the activation functions maps, scaling contributes no additional value.
- The initial parameter values for the iterative learning procedures do not depend on the input variables. If the variables are not scaled and the initial weights are not sufficiently small, the output of the bounded activation function will always be on the upper or lower bound of the range of values. In this case, the activation function has only a switching effect, similar to a threshold function.

Variables can be scaled in several ways. One possibility is to scale the data on the value range of the activation functions. This can be executed by the Min- Max- method with $x_{t_{min}}$ as the minimum and $x_{t_{max}}$ as the maximum element of one input series of length T with elements $x_t \forall t = 1, \dots, T$. According to El Ayeche and Trabelsi (2007) p.209 the scaled data on $[0; 1]$ are calculated by

$$x'_t = \frac{x_t - x_{t_{min}}}{x_{t_{max}} - x_{t_{min}}}. \quad (2.2.15)$$

The scaled data on $[-1; 1]$ result from

$$x'_t = \frac{2x_t - x_{t_{max}} - x_{t_{min}}}{x_{t_{max}} - x_{t_{min}}}. \quad (2.2.16)$$

Anders (1997) p.24 proposes to transform the data by subtracting the mean and division by the standard deviation:

$$x'_t = \frac{x_t - \bar{x}_t}{\sigma_{x_t}} \quad (2.2.17)$$

\bar{x}_t is the arithmetic mean of the values of x_t . σ_{x_t} is the square root of the variance of x_t respective \bar{x}_t . The values scaled by formula (2.2.17) should have zero mean and a standard deviation equal to one. In this case the range of values of the scaled variables is not necessarily identical with that of the activation function. However, scaling is in no way necessary. Transforming the series may lead to a loss of information (in the

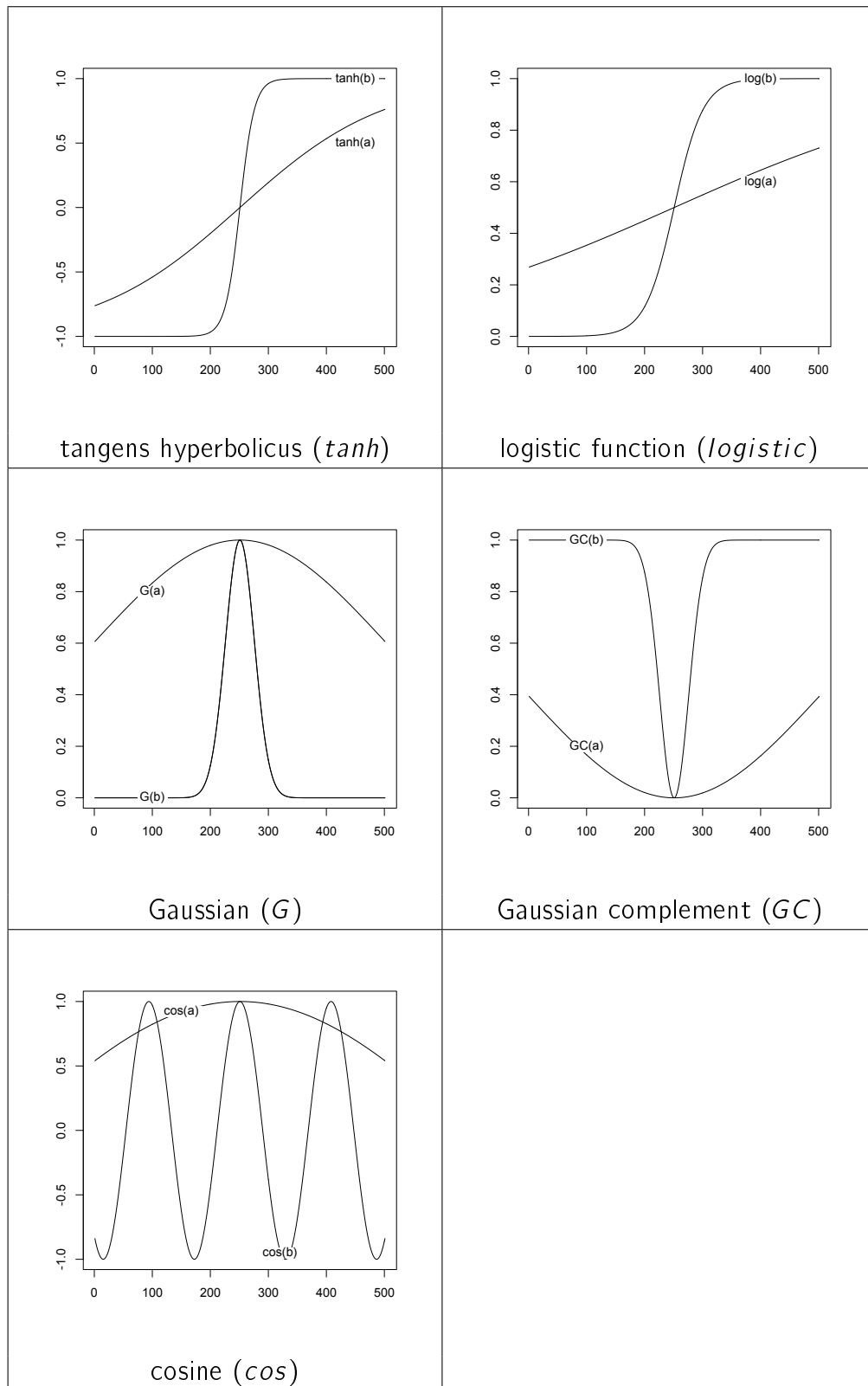


Figure 2.4: Reaction of certain activation functions on their input range

Source: Authors' design

sense of Granger and Newbold (1974)).

In figure 2.4 a linear series of 501 observations ($T = 501$) equally distributed on the intervals (a) $[-1; 1]$ and (b) $[-10; 10]$ is transformed by the mentioned bounded activation functions to visualize their behavior concerning the input range. It is observable that the larger the input is, the stronger the activation functions reacts. Note that the size of the input is not only determined by the input neurons but also by the weights. Only those bounded functions which are used as activation functions for neural networks in literature are described in this section. Thus, this section does not claim to be a perfect list of all possible bounded activation functions. However it should be mentioned here that the universal approximation property does not depend on the specific form of the activation function.

A sigmoid activation function can be interpreted as a smooth transition function, which is especially able to handle structural breaks. A closely related method to AR-NN's with sigmoid activation function is the smooth transition autoregression model (STAR). A simple version of a STAR(1) is (similar to Granger and Teräsvirta (1993) p.39):

$$x_t = \alpha_0 + \alpha_1 x_{t-1} + \Psi(\gamma_0 + \gamma_1 x_{t-1}) \beta(x_{t-1}) + \varepsilon_t, \quad (2.2.18)$$

whereas $\Psi(\cdot)$ is for example the *tanh*. This equation can be interpreted as a linear AR(1) with a structural break in the regression coefficient. The transition from regression coefficient α_1 to $\alpha_1 + \beta$ proceeds "smoothly" along the *tanh* function (an alternative would be a threshold function, which directly shifts from one model to the other). An AR-NN(1) with $h = 1$,

$$x_t = \alpha_0 + \alpha_1 x_{t-1} + \Psi(\gamma_0 + \gamma_1 x_{t-1}) \beta + \varepsilon_t, \quad (2.2.19)$$

can be interpreted as a STAR with structural break in the constant. Nevertheless equation (2.2.18) can be approximated by an AR-NN(1) with sufficient hidden neurons, as the effect of $\beta(x_{t-1})$ in (2.2.18) can be approximated by the combination of several constants. To illustrate this we consider a simple model of an AR(1) with structural break in the regression coefficient as shown in figure 2.5. In this case - for simplification

- $\Psi(\cdot)$ is a threshold function. This graph shows the following transition autoregression (TAR) process:

$$x_t = \alpha_0 + \alpha_1 x_{t-1} + \Psi(\gamma_1 x_{t-1}) \beta (x_{t-1} - 2) \quad (2.2.20)$$

with

$$\Psi(x) = \begin{cases} 1 & \text{if } x \geq 1 \\ 0 & \text{if } x < 1 \end{cases} \quad (2.2.21)$$

and $\alpha_0 = 1$, $\alpha_1 = 0.5$, $\gamma_1 = 0.5$ and $\beta = 0.5$. Now we consider an AR-NN(1) with 2 hidden neurons and function (2.2.21) as activation function:

$$\hat{x}_t = \alpha_0 + \alpha_1 x_{t-1} + \Psi(\gamma_1 x_{t-1}) \beta_1 + \Psi(\gamma_2 x_{t-1}) \beta_2 \quad (2.2.22)$$

with $\alpha_0 = 1$, $\alpha_1 = 0.5$, $\gamma_1 = \frac{1}{3}$, $\beta_1 = 1$, $\gamma_2 = \frac{1}{5}$ and $\beta_2 = 1$. Figure 2.6 shows how the AR-NN(1) in equation (2.2.22) approximates the TAR(1) of equation (2.2.20). If the number of hidden neurons is increased, the approximation becomes more accurate. To demonstrate this, we use 4 hidden neurons such that the AR-NN(1) equation becomes

$$\hat{x}_t = \alpha_0 + \alpha_1 x_{t-1} + \sum_{i=1}^4 \Psi(\gamma_i x_{t-1}) \beta_i \quad (2.2.23)$$

with $\alpha_0 = 1$, $\alpha_1 = 0.5$, $\gamma_1 = \frac{1}{2}$, $\gamma_2 = \frac{1}{3}$, $\gamma_3 = \frac{1}{4}$, $\gamma_4 = \frac{1}{5}$, $\beta_1 = 0.25$, $\beta_2 = 0.5$, $\beta_3 = 0.5$ and $\beta_4 = 0.5$. The result is shown in figure 2.7. A structural break in the regression coefficient can be approximated by a sufficiently large number of structural breaks in the constant (which are represented by the hidden neurons in an AR-NN). This simple example shows the advantage of AR-NN: The number of hidden neurons can be increased until an optimal approximation is reached. Concerning prediction, the AR-NN only delivers appropriate results in the short run. Consider figure 2.8: The larger the prediction horizon becomes, the more the original and the estimated series diverge and the more the prediction error increases. Our empirical results also confirm the finding that AR-NN perform well mainly in one and two step predictions: For higher step predictions they are dominated by their linear part.

In the further proceedings we will use often the *tanh* activation function for the following two reasons: Firstly, it is one of the most common activation functions in literature, and secondly, its derivations can be calculated relatively easily and thus it is easy to handle.

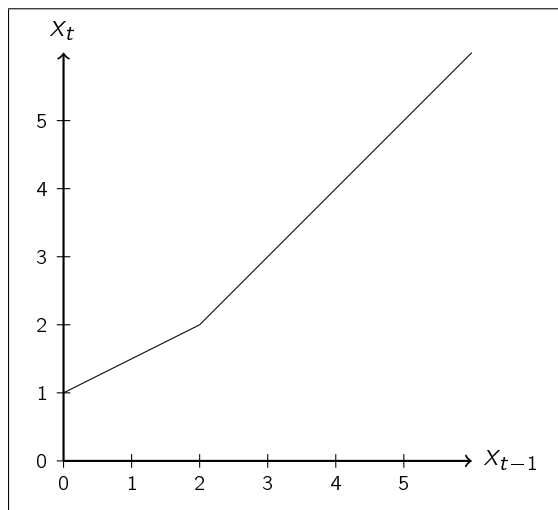
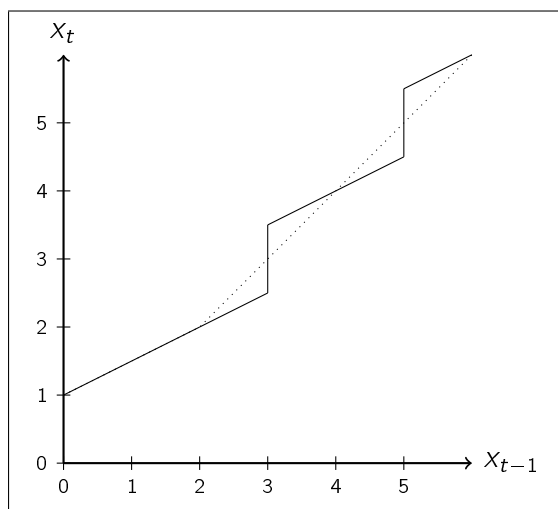


Figure 2.5: AR(1) with structural break

Source: Authors' design

Figure 2.6: AR-NN(1) with $h=2$ approximates a TAR(1)

Source: Authors' design

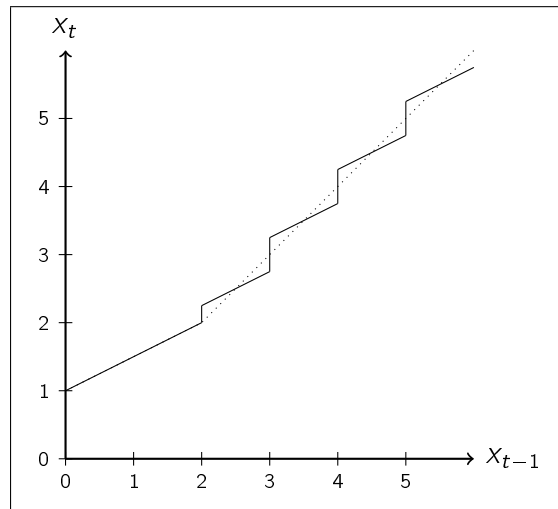


Figure 2.7: AR-NN(1) with $h=4$ approximates a TAR(1)

Source: Authors' design

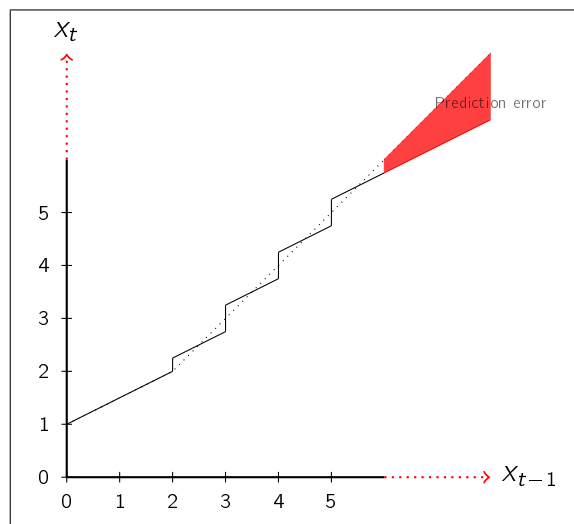


Figure 2.8: Prediction with the model from figure 2.7

Source: Authors' design

2.3 Stationarity of AR-NN

Before the “classical” scheme of Box and Jenkins (1976) part II (consisting of variable selection, parameter estimation and model validation) can be applied to the time series, it has to be tested for stationarity and eventually preprocessed to a stationary representation (usually by differentiation). This section begins with a general definition of stationarity and shows why weak stationarity is sufficient in the case of Gaussian white noise errors. Furthermore the important findings of Trapletti, Leisch and Hornik (2000) concerning stationarity tests in AR-NN are introduced. They say that the popular linear unit root stationarity tests are sufficient if the activation function is bounded. We give a short introduction into the principle of unit root tests and focus on a modification of the the Augmented-Dickey-Fuller test (ADF) for nonlinear environments, the Rank-ADF test of Hallman (1990). This test can be used as an ex-ante stationarity test, especially for nonlinear time series, as most modelling procedures require stationary data. Of course tests other than the RADF are possible, but it is simple to implement and based on the ADF test, the most common unit root test in econometrics.

2.3.1 Stationarity and Memory

A generalization of the concept of stationarity, which shows the idea behind it, can be found in Granger and Teräsvirta (1993) p.51. We introduce this first to examine the role which the information contained in the lagged values of x_t plays in explaining the long run behavior. This so called memory-concept is the information theoretic basis from which we later attach to the concept of stationarity, which we define particularly for processes with normal distributed errors (as only the first and second moment are used to describe the distribution). Let x_{t+h} be the h step forecast and Inf_t be the information set, which is in the case of an AR(n) given by $Inf_t : X_{t-1}$. The conditional expectation belonging to x_{t+h} given the information set is $E(x_{t+h}|Inf_t)$. If the expectation of the stochastic part is zero, $G(\Theta, X_{t-1})$ is an optimal estimator for $E(x_{t+h}|Inf_t)$ in the sense of the mean square principle (Leisch, Trapletti and Hornik (1998) p.2). We say the process x_t is short memory in mean (SMM) if

$$\lim_{h \rightarrow \infty} E(x_{t+h}|Inf_t) = c \quad (2.3.1)$$

and the distribution of the random variable c does not depend on Inf_t . In the special case of mean-stationarity as we will see in definition 2.2, a constant mean is a special case of SMM. In contrast, if the distribution of c depends on Inf_t , the process x_t is called long memory in mean (LMM).

We now consider the conditional distribution of the h step forecast expressed by the probability $Prob(x_{t+h} \geq x | Inf_t)$. If the limit of this conditional distribution,

$$\lim_{h \rightarrow \infty} Prob(x_{t+h} \geq x | Inf_t) \quad (2.3.2)$$

does not depend on Inf_t , the process x_t is said to be short memory in distribution (SMD). Just another notation for this would be if for all sets \mathcal{C}_1 and \mathcal{C}_2 holds

$$|Prob(x_{t+h} \in \mathcal{C}_1 | Inf_t \in \mathcal{C}_2) - Prob(x_{t+h} \in \mathcal{C}_1)| \xrightarrow{h \rightarrow \infty} 0 \quad (2.3.3)$$

If in contrast (2.3.2) depends on Inf_t , the process is called long memory in distribution (LMD). In the case of a stationary $AR(n)$ the distribution of the process is determined by ε_t which is by definition 2.1 i.i.d. $N(0, \sigma^2)$ with constant σ^2 . Thus a stationary process is SMD. The property SMD implies also the property SMM but Granger and Teräsvirta (1993) pp.51-52 provide some examples that this relation does not work in the other direction.

Stronger than the term SMD would be the term stationary in distribution, which means that (2.3.2) is constant. According to Leisch, Trapletti and Hornik (1998) p.2 this also can be called strict stationarity as it includes of course stationarity in mean. The term weak stationarity as we will define it in the following, according to Schlittgen and Streitberg (1995) p.100 and Hamilton (1994) p.45, is included in the definition of strict stationarity. Weak stationary means that only the first and second moment have to be stationary. Particularly for normal distributed processes weakly stationarity can be used synonymously for strict stationarity as the distribution is mainly characterized by the first and second moment (this is intuitively clear if one considers the Gaussian probability density function).

Definition 2.2 (Stationarity):

A stochastic process x_t is called

- **Mean-Stationary** if $E(x_t) = \text{constant} \forall t \in T$
- **Variance-Stationary** if $\sigma_t^2 = \sigma^2 = \text{constant} \forall t \in T$
- **Covariance-Stationary** if $\text{cov}(x_{t-i}, x_{t-j}) = \text{constant} \forall t \in T$ and $i, j = 0, \dots, n$
- **Weakly Stationary** if it is mean-stationary and covariance-stationary

Remark 2.2.1:

If a process is covariance-stationary the covariance of any two lag variables depends only on the distance between the lags. Clearly the i.i.d. $N(0, \sigma^2)$ process ε_t is stationary as $E(\varepsilon_t) = 0 \forall t$, $\sigma_t^2 = \sigma^2 \forall t$ and

$$\text{cov}(x_{t-i}, x_{t-j}) = \begin{cases} 0 & \text{if } i \neq j \\ \sigma^2 & \text{if } i = j \end{cases}. \quad (2.3.4)$$

Remark 2.2.2:

Covariance stationarity implies variance stationarity as $\text{cov}(x_{t-i}, x_{t-i}) = \text{cov}(x_{t-j}, x_{t-j}) = \sigma^2$.

2.3.2 Markov Chain Representation and the Invariance Measure

For the further procedure we need a function representing equation (2.2.8) which maps from $\mathbb{R}^n \rightarrow \mathbb{R}^n$. We get it by the Markov chain representation

$$X_t = H(X_{t-1}) + E_t. \quad (2.3.5)$$

The vectors belonging to this equation are $X_t = (x_t, x_{t-1}, \dots, x_{t-n+1})^\top$, $H(X_{t-1}) = (G(\Theta, X_{t-1}), x_{t-1}, \dots, x_{t-n+1})^\top$ and $E_t = (\varepsilon_t, 0, \dots, 0)^\top$ (see Trapletti, Leisch and Hornik (2000) p.2429). A Markov chain resembles a multivariate AR(1). Markov chain theory provides some additional measures to analyze the stability (for a detailed introduction see Haigh (2010) pp.88-89). Our aim is to use those stability measures for formulating a theorem concerning the stability of AR-NN (theorem 2.2). Equation

(2.3.5) provides the link between the measures from Markov chain theory and AR-NN(n) with $n > 1$.

Again we need the term SMD. As we have seen that SMD includes SMM, a Markov chain which is SMD is also stationary (see Resnick (1992) p.116). To show under which conditions a Markov chain like equation (2.3.5) is strictly stationary (and thus weakly stationary in the case of Gaussian WN errors) we need a term for the probability that x_t moves from point x to a set \mathcal{A} in k steps, denoted by $Prob^k(x, \mathcal{A}) = Prob(x_{t+k} \in \mathcal{A} | x_t = x)$. This probability is called the k -step transition probability (Fonseca and Tweedie (2002) p.651). If this transition probability is constant for all steps k we have in fact a strictly stationary process. Account for the fact that the Markov chain (2.3.5) is a AR(1), then the transition probability is equal to the first probability term in (2.3.3). Let $\|\cdot\|$ be the total deviation norm. If a constant probability measure dependent on the selection of \mathcal{A} , $\pi(\mathcal{A})$, exists such that

$$\lim_{k \rightarrow \infty} \rho^k \|Prob^k(x, \mathcal{A}) - \pi(\mathcal{A})\| = 0, \quad (2.3.6)$$

the process is called geometrical ergodic and ergodic for the special case $\rho = 1$. The probability measure has to satisfy the invariance equation

$$\pi(\mathcal{A}) = \int_{\mathbb{R}^n} Prob(x, \mathcal{A}) \pi dx. \quad (2.3.7)$$

π is also called the stationary or invariant measure. Geometrical ergodicity implies stationarity as the distribution converges to π , which is constant. Thus a geometrical ergodic process is asymptotic stationary (because of the convergence). If a process already starts with π , it is strictly stationary. In addition we need the properties irreducible and aperiodic. Irreducible can be explained informally as the property that any point of the state space of the process can be reached independently from the starting point. The process is aperiodic, if it is not possible that the process returns to certain sets only at certain time points. If the errors in our process are i.i.d. $N(0, \sigma^2)$, it is certainly irreducible and aperiodic (see Trapletti, Leisch and Hornik (2000) p.2431). Hence we will not further discuss those terms as they are included in the Gaussian WN assumption on the errors.

2.3.3 Unit Roots and Stationarity of AR-NN

This section is mainly based on Trapletti, Leisch and Hornik (2000) p.2431. For the further proceeding we first introduce some notations from linear time series analysis: The unit roots (UR) of the characteristic polynomial. Consider the scalar linear AR(n)

$$x_t = \alpha_1 x_{t-1} + \alpha_2 x_{t-2} + \dots + \alpha_n x_{t-n} + \varepsilon_t \quad (2.3.8)$$

$$\varepsilon_t = x_t - \alpha_1 x_{t-1} - \alpha_2 x_{t-2} - \dots - \alpha_n x_{t-n} \quad (2.3.9)$$

The characteristic polynomial of this process is denoted by

$$1 - \alpha_1 z - \alpha_2 z^2 - \dots - \alpha_n z^n = 0, \quad (2.3.10)$$

see Schlittgen and Streitberg (1995) p.100. The solutions z of this equation are called roots. The process is weakly stationary if the roots are outside the unit circle and thus $|z| > 1$ and explosive or chaotic if $|z| < 1$, (Hatanaka (1996) p.22). A condition equivalent to the condition that the roots should be outside the unit circle is $|\alpha_i| < 1$ (see Schlittgen and Streitberg (1995) pp.123-124 and Hatanaka (1996) pp.22-23).

If the process has its roots outside the unit circle, it can be inverted to an infinite MA representation based on the residuals. In this case it can easily be shown that x_t is stationary, because it only depends on the white-noise process ε_t . Therefore we rewrite equation (2.3.8) using the lag-operator L :

$$x_t = \alpha(L)x_t + \varepsilon_t = (1 - \alpha_1 L - \dots - \alpha_n L^n)x_t + \varepsilon_t. \quad (2.3.11)$$

The process has an infinite MA representation if the inverse filter $\alpha^{-1}(L)$ exists. Therefore with equation (2.3.11) becomes

$$x_t = \alpha^{-1}(L)\varepsilon_t = \sum_{i=1}^{\infty} \alpha^i \varepsilon_{t-i}. \quad (2.3.12)$$

The inverse filter exists only if $|z| < 1$, see Schlittgen and Streitberg (1995) p.122 and Hassler (2007) p.48.

In the border case the largest solution is $|z| = 1$. Equation (2.3.10) becomes

$$1 = \alpha_1 + \alpha_2 + \dots + \alpha_n. \quad (2.3.13)$$

We say the process has a unit root. This process can be stationarized by differentiation, because the stable filter $(1 - L)$ can be splitted off from $\alpha(L)$. Without stationarization via differences the process has no $MA(\infty)$ representation and is not stationary. A nonstationary process with one UR is called process of integration order 1. An important theorem concerning the stationarity of an AR-NN can be formulated according to Trapletti, Leisch and Hornik (2000) pp. 2430-2431:

Theorem 2.2 (Trapletti, Leisch and Hornik (2000) pp. 2430-2431 theorem 1):

Assume that ε_t is a Gaussian WN process and Ψ is bounded. The characteristic polynomial of the linear part (the direct edges between input and output nodes without the bias) is denoted as

$$\alpha(z) = 1 - \sum_{i=1}^n \alpha_i z^i. \quad (2.3.14)$$

The condition

$$\alpha(z) \neq 0 \quad \forall z, \quad |z| \geq 1, \quad (2.3.15)$$

is sufficient but not necessary that the process x_t is geometrical ergodic and asymptotic stationary. If $E|\varepsilon_t|^2 < \infty$, the process is weakly stationary.

PROOF: The proof can be formulated in two different ways using two previous findings: The first proof in Trapletti, Leisch and Hornik (2000) p. 2438 uses the results of Tjøstheim (1990) and Meyn and Tweedie (1993). The alternative, the proof of theorem 1 (Leisch, Trapletti and Hornik (1998) p.4) in Leisch, Trapletti and Hornik (1998) pp. 9-10 uses the results of Chan and Tong (1985).

The bias is processed like a constant and is thus not part of the characteristic polynomial (like deterministic drifts in linear $AR(n)$). We see that stationarity of the process depends on the linear part and we can use the usual unit root theory from linear time series analysis to test for stationarity. If we have no linear part, the AR-NN always leads to a stationary representation (because of the boundedness of the activation function),

see Trapletti, Leisch and Hornik (2000) p.4.

Next it has to be shortly explained why (weak) stationarity of the linear part is not a necessary condition. If one root is on the unit circle, we expect Random Walk behavior of the process with or without time trend (drift). But it is possible that the nonlinear part of the process causes a drift towards a stationary solution. This is meant by the statement that stationarity of the linear part is sufficient but not necessary in theorem 2.2. For further details see Trapletti, Leisch and Hornik (2000) p.2432. Theorem 2.2 outlines also the hybrid character (composed of a linear and a nonlinear part) of AR-NN as we use them. Practical application especially with unscaled data shows that the squashing property for bounded activation functions tends to produce stationary outputs.

2.3.4 The Rank Augmented Dickey-Fuller Test

In this subsection we consider a procedure based on the ADF test. This test is most common in econometrics to analyze the UR of a given time series (originally developed by Dickey and Fuller (1979) for linear AR(1)). Dickey-Fuller tests for higher order linear AR are called ADF tests, see Schlittgen and Streitberg (1995) p.300. In the following we give a short overview over the test and consider some problems using UR tests.

At first the linear AR with unknown order n of equation (2.1.2) is rearranged:

$$x_t = a_1 x_{t-1} + a_2 \Delta x_{t-1} + a_3 \Delta x_{t-2} + \dots + a_n \Delta x_{t-n} + \varepsilon_t \quad (2.3.16)$$

whereas $a_1 = \alpha_1 + \alpha_2 + \dots + \alpha_n$, $a_2 = -\alpha_2 - \dots - \alpha_n$, $a_3 = -\alpha_3 - \dots - \alpha_n$, $a_n = -\alpha_n$ and ε_t are the residuals of the equation. The null hypothesis " $H_0 : a_1 = 1$ " implies that the process is integrated of order one which means it can be stationarized by applying first differences. The alternative is " $H_1 : a_1 < 1$ ", which means that the process is already stationary. Usually the following test statistic is applied:

$$T_{ADF} = \frac{a_1 - 1}{\sqrt{\sigma_{\varepsilon_t}^2 \cdot \sum_{t=2}^T x_{t-1}^2}}. \quad (2.3.17)$$

$\sigma_{\varepsilon_t}^2$ (the variance of the residuals) and a_1 are received by linear regression applied at equation (2.3.16), see Schlittgen and Streitberg (1995) p.300. The distribution of T_{ADF}

in a linear environment is tabulated for example in Fuller (1976) p.373. Hallman (1990) pp. 7-58⁴ considers nonlinear transformations of linear time series. He found out that an ADF test using the ranks of a time series works better, if the underlying dynamic of the series is nonlinear (see Hallman (1990) p.43). The rank of a single observation x_t of a time series, $R(x_t)$, is defined as the rank of x_t in the ordered time series (see Hallman (1990) p.34). Thus for computation of the ranks of a time series in a first step the observations are increasingly ordered by their value. In a second step a rank corresponding to the place in the ordered series is assigned to each element of the originally unordered series. Equation (2.3.16) for the ranks of the series is

$$R(x_t) = a_1 R(x_{t-1}) + a_2 \Delta R(x_{t-1}) + a_3 \Delta R(x_{t-2}) + \dots + a_n \Delta R(x_{t-n}) + \varepsilon_t. \quad (2.3.18)$$

The test statistic is calculated analogous to (2.3.17). For the rank- ADF (RADF) new critical values are necessary, which are provided in Hallman (1990) p.39. Table 2.3 shows them for models without constant.⁵ A series is considered to be stationary, if the test statistic is below the critical value. If the RADF test indicates that a series is nonstationary, differences of the ranks have to be used rather than ranks of the differences to keep the procedure in accordance with the ADF test.

| T | 10% | 5% | 1% |
|----------|------------|-----------|-----------|
| 25 | -1.67 | -2.05 | -2.87 |
| 50 | -1.57 | -1.91 | -2.56 |
| 100 | -1.61 | -1.92 | -2.52 |
| 200 | -1.66 | -1.95 | -2.57 |
| 400 | -1.70 | -2.04 | -2.61 |
| 800 | -1.79 | -2.08 | -2.73 |

Table 2.3: RADF critical values (Hallman (1990) p.39)

Linear regression for calculation of the test statistics (out of equation (2.3.16)) requires specification of the lag order. As we will see in the following section, lag selection procedures require on the other hand stationary data. This leads to the dilemma that the one is not possible without the results of the other. Said and Dickey (1984) propose to estimate the AR-order as a function of T . In our opinion this does not completely solve the problem as the UR-test should be based on the same lag order as the estimated

⁴This chapter of Hallman (1990) was also published in a shorter version as Granger and Hallman (1988)

⁵Note that the presign is changed, because in Hallman (1990) p.29 the presign of the test statistic is different

model. Therefore a good strategy might be to define a maximal lag order m at first and then calculate the RADF test statistic for all lags from 1 to m . The maximal lag order should account for the fact that a realistic relation between n and T is kept. In Said and Dickey (1984) p.600 the optimal relation between lags and observations is determined by ${}^3\sqrt{T} \cdot m \rightarrow 0$ for example. Another possibility is to prespecify the number of hidden neurons and to determine a maximal lag order which accounts for the fact that the relation between parameters and observations should be 1/10. For example if we have $h = 2$ hidden neurons, the maximal number of lags should be $m \approx 1/30T + 5/3$. Usually we expect to find for each lag order $n = 1, \dots, m$ that the process is nonstationary and can be stationarized by the first differences. Subsequently one of the several variable selection procedures from the following section can be applied to the stationarized data to find the optimal lag order n for the models.

3 Modelling Univariate AR-NN

In this chapter we show how one belongs to an univariate AR-NN model for a given time series. Only estimating the parameters is certainly not sufficient to receive an appropriate model. We follow Box and Jenkins (1976) part II, who propose to proceed in three steps: Variable selection, parameter estimation and model validation (parameter tests). Before we can start with the first step, it has to be assured that the data are stationary. If they are nonstationary, stationarization as an additional transformation is necessary. The aim of doing so is to avoid the problem of spurious regression mentioned by Granger and Newbold (1974), which may occur if regression of any nonstationary time series which are in reality uncorrelated on each other indicates significantly correlated results. Although this problem is analyzed only for linear time series, several authors state that it is relevant for neural networks too (Lee, White and Granger (1993) p.287, Anders (1997) p.99, Trapletti, Leisch and Hornik (2000) p.2440). Steurer (1996) p.120-124 shows by empirical investigation that neural networks only work accurate for stationary data. Hence the methods presented below are only applicable at stationary time series. For testing procedures for stationarity we refer to section 2.3.4.

Nonlinearity tests are an addition to the common framework of time series modelling. Testing a time series on hidden nonlinearity before an AR-NN model is adjusted is necessary for two reasons: Firstly the additional effort necessary for a nonlinear model compared to a linear one has to be justified. Secondly as we know from section 2.2.3 an AR-NN model performs equal to a linear model if the investigated time series is determined by a linear process. If one accounts for the additional effort necessary for the nonsignificant linear part in such a case, the AR-NN is inferior than a simple linear AR.

In figure 3.1 a flow chart shows the steps to build an AR-NN model for a given time series. This figure may serve as a general plan to construct an AR-NN model of any given time series as also the “augmented” philosophy is included (increasing the number of hidden neurons step by step, starting with $h=1$). The figure also serves as a guideline for the sections of this chapter.

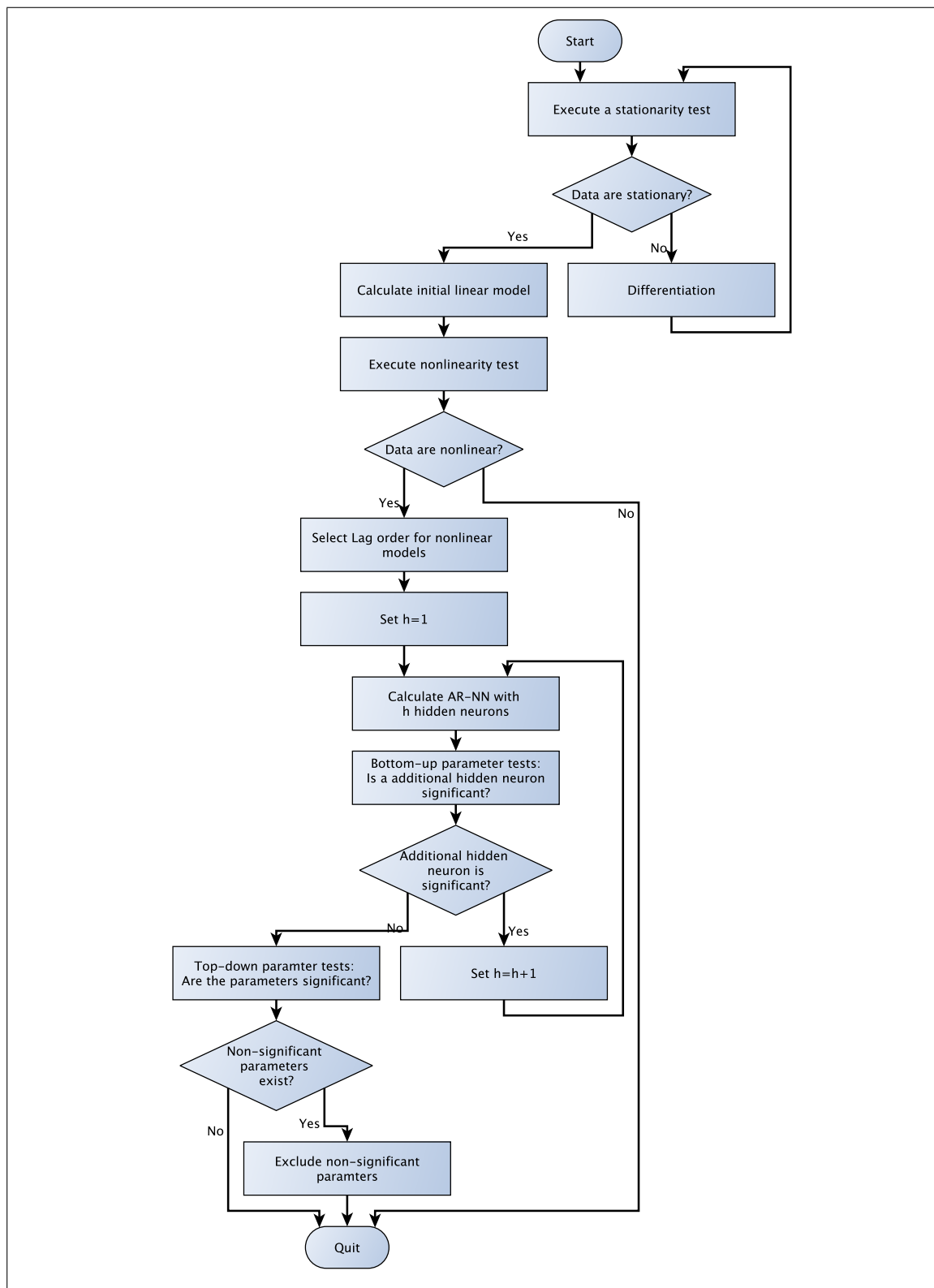


Figure 3.1: Flow chart AR-NN model building

Source: Authors' design, based on the figures in Anders (1997) p.37 and pp.127-132

3.1 The Nonlinearity Test

In the previous sections we have become acquainted with the structure of AR-NN. Compared to a simple linear model it is much more complicated. If a series is linear, the additional effort is of no use. To avoid this, the series should be tested on hidden nonlinearity at the first. The nonlinearity test of Teräsvirta, Lin and Granger (1993), described in this section, is a simple and efficient method, based on previous results of White (1989a). The basic idea is to approximate the AR-NN by a Taylor polynomial. Such an approximation is necessary, because the distribution of the parameters of the AR-NN does not always exist (only if the conditions in section 3.4.2.1 are achieved). In contrast the distribution of the parameters of the Taylor polynomial always exists. An existing distribution of the parameters is a prerequisite for parameter tests like the Lagrange-multiplier test. As we have not yet specified the number of lags, the test on hidden nonlinearity should be executed on all lags from 1 to a prespecified maximal number of lags. Empirical application shows, that a time series may be nonlinear for one lag order as well as linear for other lag orders.

3.1.1 Taylor Expansion

Taylor expansion is a method to approximate nonlinear functions by a chain of polynomials of increasing order. The concept is easy to understand, hence this approximation method will be of use several times in this dissertation. Its two main advantages are its general approximation property and the existence of a distribution of its parameters and therefore its adaptability for parameter tests. We specify it in the following:

Based on Weierstrass (1885) and extended by Stone (1948), the Stone-Weierstrass theorem says that a Taylor polynomial of sufficient high order can approximate any function (see Medeiros, Teräsvirta and Rech (2006) pp.52-53). Taylor expansion of order k for a function $F(x)$ around a point x_0 is given by

$$F(x) = F(x_0) + \frac{F'(x_0)}{1!}(x - x_0) + \frac{F''(x_0)}{2!}(x - x_0)^2 + \dots + \frac{F^k(x_0)}{k!}(x - x_0)^k, \quad (3.1.1)$$

with $F'(x_0)$ as the first derivative of $F(x_0)$ respective x_0 , $F''(x_0)$ the second derivative and $F^k(x_0)$ the k th derivative, see Anders (1997) p.52. If $x_0 = 0$ (Taylor expansion around 0) the series (3.1.1) is also called Maclaurin series. If our aim is to approximate

an unknown nonlinear function $F(X_{t-1})$ in equation (2.1.1) we can not determine $F(0)$ as well as the derivatives $F'(0), F''(0), \dots, F^k(0)$. In general the derivatives with respect to zero consist only of constant parts. All constants in a Maclaurin series can be combined in parameters ϕ such that it is no longer necessary to know the derivatives. Therewith equation (3.1.1) can be written as a function only consisting of parameters and variables. With a polynomial approximation of $F(X_{t-1})$ equation (2.1.1) becomes

$$\begin{aligned}
 x_t = & \phi_0 + \underbrace{\sum_{j_1=1}^n \phi_{j_1} X_{t-j_1}}_{\text{linear component}} + \underbrace{\sum_{j_1=1}^n \sum_{j_2=j_1}^n \phi_{j_1, j_2} X_{t-j_1} X_{t-j_2}}_{\text{quadratic component}} \\
 & + \underbrace{\sum_{j_1=1}^n \sum_{j_2=j_1}^n \sum_{j_3=j_2}^n \phi_{j_1, j_2, j_3} X_{t-j_1} X_{t-j_2} X_{t-j_3}}_{\text{cubic component}} + \dots + \\
 & \underbrace{\sum_{j_1=1}^n \dots \sum_{j_k=j_{k-1}}^n \phi_{j_1, j_2, \dots, j_k} X_{t-j_1} \dots X_{t-j_k}}_{\text{k component}} + u_t
 \end{aligned} \tag{3.1.2}$$

whereas u_t is the residual part consisting of ε_t plus the additional error caused by the approximation. The number of parameters is $m(k) = \frac{(n+k)!}{n!k!}$.

The number of parameters increases with k or n . This means that for models with high lag orders even low order polynomials include an immense number of parameters (for example if $n = 15$ and $k = 3$, $m(k) = 816$ parameters have to be estimated). The dilemma is now that if one wants to increase the precision of the Taylor polynomial by increasing k , also the number of parameters increases multiplicatively. Simple estimation procedures like ordinary least squares (OLS) can only identify the parameters for a limited number of lags which should be small compared to the number of observations T (see section 3.2.3 for more about this). The main problem in polynomial approximation is a conflict of objectives between precision and avoiding overparametrization. Nevertheless polynomial approximation is an easy concept which may produce quite acceptable results. Note that like in linear autoregression the data should be stationarized to avoid spurious regression. For an implementation of an OLS estimation of equation (3.1.2) see section 3.2.3.

To reduce the complexity we take up a parametric position as we already have specified

the nonlinear function (here the \tanh). The following procedure partly follows the ideas of Granger and Lin (1994). Instead of using the complex equation (3.1.2) we may use equation (3.1.1) to specify the polynomial as the structure of $F(X_{t-1})$ is known (the \tanh) and we are able to calculate the derivatives. We assume that the order of the polynomial $k = 3$ is sufficient (for example $k = 3$ is used by Granger and Lin (1994)). The first, second and third order derivations of $\tanh(0)$ are (see Anders (1997) p.53):

$$\begin{aligned}\tanh'(0) &= 1 \\ \tanh''(0) &= 0 \\ \tanh'''(0) &= -2\end{aligned}$$

Using those results equation (3.1.1) with $k = 3$ becomes

$$x_t = \phi_0 + \sum_{j_1=1}^n \phi_{j_1} x_{t-j_1} - \frac{1}{3} \sum_{j_1=1}^n \sum_{j_2=j_1}^n \sum_{j_3=j_2}^n \phi_{j_1, j_2, j_3} x_{t-j_1} x_{t-j_2} x_{t-j_3} + u_t \quad (3.1.3)$$

The number of parameters is reduced to $m(3)^* = n + 1 + \frac{(n+3-1)!}{(n-1)!3!}$ compared to $m(3) = \frac{(n+3)!}{n!3!}$ if equation (3.1.2) is used. To illustrate this advantage let $n = 6$. Using polynomial (3.1.2) requires the estimation of $m(3) = 84$ parameters, polynomial (3.1.3) in contrast requires only $m(3)^* = 63$ parameters.

The approximation property of the function in equation (3.1.3) can be illustrated by the following example (similar to Anders (1997) p.53): Let x be a linear increasing series in the interval $[-5; 5]$ with 80 observations and let

$$y = \tanh(x). \quad (3.1.4)$$

This function is plotted in figure 3.2 (black line). The red line in figure 3.2 shows the polynomial approximation of equation (3.1.4),

$$\hat{y} = \phi_0 + \phi_1 x - \frac{1}{3} \phi_2 x^3. \quad (3.1.5)$$

This equation can be estimated by an OLS procedure. We see that the polynomial is able to approximate the original tanh quite well, but the regression in the "transition" part is more flat.¹

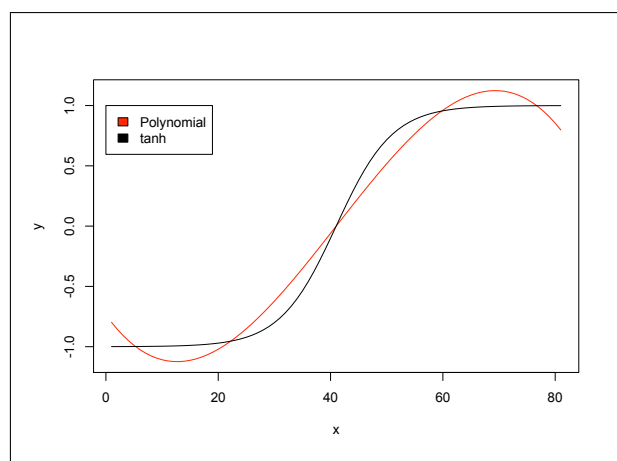


Figure 3.2: Taylor polynomial approximation of the tanh

Source: Authors' design, based on Anders (1997) p.53 figure 3.5

3.1.2 The Lagrange-Multiplier Tests

The test of Teräsvirta, Lin and Granger (1993) is an advancement of the test of White (1989a). Both tests are executed as Lagrange-multiplier (LM) tests. The main difference is, that in the test of White (1989a) weights for the additional hidden neurons are determined randomly, whereas in the test of Teräsvirta, Lin and Granger (1993) the nonlinear part is approximated by Taylor expansion. The main difficulty with the test of White (1989a) is the problem of inconsistency in the case of heteroscedasticity in calculation of the test statistic. Another problem is the arbitrariness in selection of the nonlinear hidden units weights. As the test of Teräsvirta, Lin and Granger (1993) tries to overcome those problems, we use it in the further procedure. The following description starts with the test of White (1989a) (see also Anders (1997) pp.69-72) in subsection 3.1.2.1 to introduce the basic principles of LM nonlinearity tests. Subsection 3.1.2.2 continues with the advancements of Teräsvirta, Lin and Granger (1993) (see also Anders (1997) pp.67-69)

¹The main difference between our figure 3.2 and the figure in Anders (1997) p.53 are the parameters; In our example they are estimated by OLS, in Anders (1997) p.53 they are set 1. The advantage of our result is, that the polynomial estimator is bounded on the right and left end of the "transition" part. However the approximation of the boundaries is not appropriate as they are not flat in the polynomial.

3.1.2.1 The Test of White

Neglected nonlinearity in the sense of White (1989a) p.45 means, that there is some nonlinearity which is not covered by the linear AR in the process. If neglected nonlinearity exists - and the process is in fact determined by a nonlinear function - the linear AR model is misspecified. Now the test should examine whether the linear model is misspecified or not. Thus the null hypothesis in the test is, that an estimated linear model

$$x_t = \alpha_0 + \alpha_1 x_{t-1} + \dots + \alpha_n x_{t-n} + \varepsilon_t = \alpha_0 + A^\top X_{t-1} + \varepsilon_t \quad (3.1.6)$$

with $X_{t-1} = (x_{t-1}, \dots, x_{t-n})^\top$ and $A = (\alpha_1, \dots, \alpha_n)^\top$ is able to explain the "true" function $F(X_{t-1})$ in equation (2.1.1). The hypothesis formally can be written as

$$H_0 : Prob(F(X_{t-1}) = \alpha_0 + A^\top X_{t-1}) = 1 \quad (3.1.7)$$

The alternative hypothesis is, that the linear AR does not explain $F(X_{t-1})$,

$$H_1 : Prob(F(X_{t-1}) = \alpha_0 + A^\top X_{t-1}) < 1. \quad (3.1.8)$$

The test of H_0 against H_1 is constructed on the basis of the WN assumption on the residuals. If H_0 does not apply and not the whole true function $F(X_{t-1})$ is explained by $\alpha_0 + A^\top X_{t-1}$, then some neglected nonlinearity is spuriously contained in the stochastic part. Consider equation (3.1.6). To separate the neglected nonlinearity from the stochastic part we must rewrite this equation as

$$x_t = \alpha_0 + A^\top X_{t-1} + u_t, \quad (3.1.9)$$

with

$$u_t = (F(X_{t-1}) - \alpha_0 - A^\top X_{t-1}) + \varepsilon_t \quad (3.1.10)$$

and ε_t is i.i.d. with $N(0, \sigma^2)$. The first term of equation (3.1.10) notes the part of $F(X_{t-1})$ which is not covered by the linear process, the neglected nonlinearity. If such is present, the residuals u_t are actually not equal to the stochastic part ε_t but contain the neglected nonlinearity in addition. If H_0 applies, the first term of equation (3.1.10) vanishes and the residual term u_t consists only of the Gaussian distributed WN part ε_t . In this case equation (3.1.9) becomes (3.1.6) and the linear estimation is appropriate.

If H_0 is true, there is no correlation between the residual term and X_{t-1} , which means that the conditional expectation $E(u_t|X_{t-1}) = E(\varepsilon_t|X_{t-1}) = 0$ (see also section 2.1.1). Hence even if X_{t-1} is transformed by any function $\mathcal{H}(X_{t-1})$, the residual term u_t is not correlated with that transformation, because

$$E(\mathcal{H}(X_{t-1}) \cdot u_t) = E(E(\mathcal{H}(X_{t-1}) \cdot u_t|X_{t-1})) = E(\mathcal{H}(X_{t-1}))E(u_t|X_{t-1}) = 0. \quad (3.1.11)$$

We define $\mathcal{H}(X_{t-1})$ as an additional hidden unit, called “phantom unit”. It can be constructed using the activation function of the network Ψ , random weights $\gamma_0, \gamma_1, \dots, \gamma_n$ and a random β . Note that in the original paper of White (1989a) more than one additional hidden unit can be used. To keep the test manageable, we propose to use only one additional unit, also in regard of a proper relation between observations T and the number of parameters. The following hypotheses are based on the AR-NN equation

$$x_t = \alpha_0 + A^\top X_{t-1} + \Psi(\gamma_0 + \Gamma^\top X_{t-1})\beta + \varepsilon_t, \quad (3.1.12)$$

which corresponds to equation 2.2.7 with $h=1$. The test of White (1989a) is based only on the γ -weights. Respecting this, a consequence of the null hypothesis H_0 (3.1.7) is

$$H_0^* : E(\Psi(\gamma_0 + \Gamma^\top X_{t-1}) \cdot u_t | \gamma_i) = 0 \quad \forall i = 0, \dots, n \quad (3.1.13)$$

with the alternative

$$H_1^* : E(\Psi(\gamma_0 + \Gamma^\top X_{t-1}) \cdot u_t | \gamma_i) \neq 0 \quad \forall i = 0, \dots, n \quad (3.1.14)$$

Thus the rejection of H_0^* means rejecting H_0 . However not rejecting H_0^* does not mean not rejecting H_0 . Consequently testing H_0^* against H_1^* is not consistent for testing H_0 against H_1 . But as a test of H_0^* against H_1^* can be implemented as a LM test, we continue with that procedure.

In order to get to the test statistic first the expectation $E((\gamma_0 + \Gamma^\top X_{t-1}) \cdot u_t | \gamma_i)$ has to be estimated, which is done by calculating the average:

$$E(\Psi(\gamma_0 + \Gamma^\top X_{t-1}) \cdot u_t | \gamma_i) = \frac{1}{T} \sum_{t=1}^T (\Psi(\gamma_0 + \Gamma^\top X_{t-1}) \cdot u_t) \quad (3.1.15)$$

If the null hypothesis is not rejected, the value of (3.1.15) should be around zero (White (1988) p.453). If in contrast the null hypothesis is rejected, the value of (3.1.15) is away from zero. Vice versa one can test if the expectation is significantly away from zero to decide if H_0^* should be rejected. Therefore the distribution of $\frac{1}{\sqrt{T}} \sum_{t=1}^T (\Psi(\gamma_0 + \Gamma^\top X_{t-1}) \cdot u_t)$ should be known. According to the central limit theorem it converges to $N(0, \sigma^{2*})$, a normal distribution with zero mean and variance σ^{2*} (denotes here the variance of the additional hidden neuron) as $T \rightarrow \infty$.² The test statistic

$$T_1 = \frac{1}{\sqrt{T}} \sum_{t=1}^T (\Psi(\gamma_0 + \Gamma^\top X_{t-1}) \cdot u_t) \cdot \frac{1}{\sigma^{2*}} \cdot \frac{1}{\sqrt{T}} \sum_{t=1}^T (\Psi(\gamma_1 + \Gamma^\top X_{t-1}) \cdot u_t) \quad (3.1.16)$$

is χ^2 distributed with one degree of freedom. The test is implemented as χ^2 -test, which means that H_0^* is rejected if it exceeds a certain percentile of the (above) χ^2 -distribution. However sometimes it might be sometimes difficult to determine σ^{2*} . Therefore the parameter estimators have to be consistent, which is only given under certain conditions (see section 3.4.2.1). Hence a procedure is used, which leads to test statistics asymptotically equivalent to 3.1.16:

First an artificial linear regression is performed:

$$u_t = \Phi_1 \nabla (\alpha_0 + A^\top X_{t-1}) + \Phi_2 (\Psi(\gamma_0 + \Gamma^\top X_{t-1})) + u_t^*, \quad (3.1.17)$$

whereas Φ_1 and Φ_2 are parameters with dimensions $(1 \times (1+n))$ and (1×1) respectively, u_t are the residuals of equation (3.1.9) and u_t^* is the residual term of the artificial linear regression. ∇ denotes the vector of partial derivatives (gradient vector) respective the input (constant and n lags of x_t). For the following LM test statistics see Anders (1997) pp.68-69. Using the residuals u_t from equation (3.1.9) we can calculate the first LM test statistic

$$T_{LM_1} = T \cdot \frac{\sum_{t=1}^T \hat{u}_t}{\sum_{t=1}^T u_t^2}. \quad (3.1.18)$$

²The variance is univariate as we add only one "phantom" hidden neuron

It is χ^2 - distributed with 1 degrees of freedom (Φ_2 is scalar) as only one additional hidden neuron is added. This test statistic is the uncentered coefficient of determination of the artificial linear regression (3.1.17) multiplied by T . According to Davidson and MacKinnon (1993) p.189 equation (3.1.18) can be stabilized by multiplication by a factor $(T - r)$ with r equal to the number of parameters in equation (3.1.6). An alternative test statistic is

$$T_{LM_2} = \frac{\left(\sum_{t=1}^T u_t^2 - \frac{\left(\sum_{t=1}^T u_t^* \right)^2}{T} \right) / 1}{\left(\sum_{t=1}^T u_t^{*2} \right) / (T - r)} \quad (3.1.19)$$

It is F - distributed with $(n + 1)$ and $(T - r)$ degrees of freedom. It has to be mentioned that the power of the LM- test eventually rises if the errors are regressed on the principal components of the terms in equation (3.1.6) rather than on the terms itself (see White (1989a) p. 454).

3.1.2.2 The Test of Teräsvirta, Lin and Granger

The problem with the null hypothesis is, that it can only be identified if the alternative H_1 applies. Teräsvirta, Lin and Granger (1993) pp. 210-211 solve this problem by using Taylor expansion around $\gamma_i = 0 \forall i = 0, \dots, n$.³ Thus the second term of equation (3.1.17) is replaced by a third order polynomial.⁴ Thus we can apply equation (3.1.1) at the additional neuron. If equation (3.1.12) is now concretized for the *tanh* activation function we may use the results from equation (3.1.3).

As the first two terms in equation (3.1.3) represent a linear relationship, testing on nonlinearity is only based on the parameters ϕ_{j_1, j_2, j_3} in the cubic terms (the third term). If we assume that all linear relationships are already contained in the linear part we can rewrite equation (3.1.17) by using equation (3.1.3) as

$$u_t = \Phi_1 \nabla (\alpha_0 + A^T X_{t-1}) - \frac{1}{3} \sum_{j_1=1}^n \sum_{j_2=j_1}^n \sum_{j_3=j_2}^n \phi_{j_1, j_2, j_3} x_{j_1} x_{j_2} x_{j_3} + u_t^* \quad (3.1.20)$$

³This test is related to Lee, White and Granger (1993), see also section 3.4.1.1

⁴Note that if the logistic activation $\Psi_{logistic}(\cdot)$ function is used we have to subtract $\frac{1}{2}$ such that $\Psi(\cdot) = \Psi_{logistic}(\cdot) - \frac{1}{2}$ because $\Psi_{logistic}(0) = \frac{1}{2}$.

However, if the activation function is unknown, the nonlinear part can be approximated using equation (3.1.2). The null hypothesis is

$$H_0 : \phi_{2_{j_1 j_2 j_3}} = 0 \quad \forall j_1, j_2, j_3 \quad (3.1.21)$$

with alternative

$$H_1 : \phi_{2_{j_1 j_2 j_3}} \neq 0 \quad \forall j_1, j_2, j_3. \quad (3.1.22)$$

The testing procedure itself is executed as a LM test in the same way as the test of White (1989a). Φ_1 is the regression coefficient like in equation (3.1.17) for the first part and $\phi_{2_{j_1 j_2 j_3}}$ are the regression coefficients for the second part (corresponds to Φ_2 in equation (3.1.17)). If the general polynomial in equation (3.1.2) is used, the degrees of freedom are $\left(\frac{(n+3)!}{3!(n)!} - n - 1 \right)$ for T_{LM_1} and for T_{LM_2} the second degrees of freedom term is $(T-r)$. If the reduced Taylor polynomial for the *tanh* function is used (equation (3.1.3)), the degrees of freedom are $\left(\frac{(n+3-1)!}{3!(n-1)!} \right)$ for T_{LM_1} (of course this applies also to the first degree of freedom in T_{LM_2}).

As input variables are multiplied with each other in the Taylor polynomial, the danger of multicollinearity in the quadratic and cubic terms exists. A solution might be the application of principal component decomposition like proposed by White (1989a) p.454. However, the nonlinearity test should give a first insight into possible nonlinear structures only. To keep the testing procedure simple, we abstain from introducing additional principal component decomposition.

3.2 Variable Selection

Now for a time series with hidden nonlinearity in at least some lags, an AR-NN can be adjusted. Aside from estimating the parameters there are still two things to decide: Selecting the lags and detecting the number of hidden units (Medeiros, Teräsvirta and Rech (2006) p.52). To keep the computational effort straightforward, the first problem is solved ex-ante. The second problem is solved by the mentioned bottom-up strategy starting with an AR-NN with $h=1$ and increasing h stepwise. In addition ex-post significance tests on the parameters, see section 3.4.2, exclude non-significant hidden units. In general the procedure of lag and parameter selection is carried out according to the Occam's razor principle, which means to prefer the simplest model from a set of models

with the same performance. In other words only those lags and parameters should be included, which significantly improve the model.

In linear time series analysis the lag order is usually detected by calculating information criteria (IC) for several lags and choosing the lag order belonging to the smallest IC. These criterions consider not only the absolute quality of the model (like the variance of the residuals which should be minimized) but also account for the amount of computational effort if the models becomes more complicated. The most common IC is the Akaike Information Criterion (AIC), which is defined as

$$AIC_1 = T \cdot \log(\sigma^2) + 2 \cdot r, \quad (3.2.1)$$

see Burnham and Anderson (2004) p.268 and Akaike (1974) p.719 or alternatively without logarithms as

$$AIC_2 = \frac{1}{T} \sum_{t=1}^T \varepsilon_t^2 + \sigma^2 \frac{2r}{T}, \quad (3.2.2)$$

see Amemiya (1980) p.344 and Anders (1997) p.78. A well known alternative is the Schwarz- Bayesian information criterion (BIC), proposed by Schwarz (1978) pp.462-463,

$$BIC = T \cdot \log(\sigma^2) + T \cdot \log(r). \quad (3.2.3)$$

For other ICs see for example Judge et al. (1984) p.862-874.

The application of those IC in nonlinear time series analysis is criticized in several works. For example Qi and Zhang (2001) show, that there is no correlation between the IC and out-of-sample forecasting performance. Tschernig and Yang (2000) argue, that using ICs for lag selection of nonlinear processes is not based on proper theory. They show by simulation that it is sometimes inefficient. Surely estimating several AR-NN with various lags and evaluating them might be a solution in theory. Yet estimation of parameters in AR-NN is an expensive procedure. The learning algorithms provide various tools to improve the search for an optimum for a given number of lags. We will see below that evaluation of estimated AR-NN with various lag orders might lead to the question if all options for optimization of the learning algorithms have been utilized. To reduce the effort as much as possible, restrictions should be introduced, like fixing the number of lags before the parameters are estimated. Then this structure of the neural

network is the basis for search for the optimal parameter values and number of hidden neurons.

In the following chapters we discuss some lag selection procedures, which have the common property not to be restricted only to neural networks but to be applicable to all kinds of nonlinear processes as they use some general nonlinear/nonparametric (Taylor polynomials and kernel regression, subsections 3.2.2, 3.2.3 and 3.2.4) methods to approximate the unknown nonlinear function. They are able to give an approximative insight into how nonlinear models would behave and which lag structure might be optimal for them. It has to be mentioned that all procedures shown below have the limitation to work only appropriate if data are stationary. We only concentrate on those methods which also manageable with much less effort than neural networks themselves. Otherwise approximation of the neural networks for lag selection would be senseless.

3.2.1 The Autocorrelation Coefficient

The very simplest procedure from a computational point of view is based on autocorrelation coefficients (AC). This measure is not restricted to linear time series and can be applied to nonlinear series as well, see for example Lin et al. (1995). The Pearson-AC between the original series x_t and an arbitrary lag x_{t-i} in general defined as

$$AC_i = AC(x_t, x_{t-i}) = \frac{cov(x_t, x_{t-i})}{\sigma_{x_t} \sigma_{x_{t-i}}}. \quad (3.2.4)$$

The values of the AC range in the interval $[-1; 1]$. Evans (2003) p.229 propose to apply this formula only at stationary series (to avoid spurious regression), although it is often applied at nonstationary series. In practical application - particularly for nonstationary real world economic and financial data series - we observe that the data in levels have a highly autocorrelated structure, which means that the AC is significant up to a high lag order. Typical for AR is the fact, that the AC for nonstationary series decreases with increasing i . The first differences of the series are characterized by nearby no autocorrelation. Hence the AC is not a good tool for identification of the lag order, because it certainly includes too many lags if applied to a nonstationary series. In contrast the AC tend to detect no autocorrelated structure - in opposite to some other lag selection procedures we will see below - if the series is stationarized by differentiation.

The partial autocorrelation coefficients (PAC) are a modification of the AC. They describe the partial correlation between the variables x_t and x_{t-i} , whereas the variables between them are kept constant (Schlittgen and Streitberg (1995) p.194). In other words the correlation between the two variables is corrected for the influence of the variables between them. The PAC for the first lag is equal to the AC of the first lag:

$$AC_1 = PAC_1. \quad (3.2.5)$$

The PAC's for larger lags, $i > 1$, are calculated by

$$PAC_i = \frac{AC_i - \sum_{j=1}^{i-1} PAC_{i-1,j} \cdot AC_{i-j}}{1 - \sum_{j=1}^{i-1} PAC_{i-1,j} \cdot AC_{i-j}}, \quad (3.2.6)$$

see Evans (2003) pp. 229- 231. The range of the PAC is the same as for the AC. If the PAC are used for lag selection, the lag where the PAC is significant is chosen as maximal lag n . If the PAC is significant for more than one lag, the lag with the largest PAC is chosen as n . A similar lag selection criterion is the AC-criterion of Huang et al. (2006) p.514. It detects the lags to be included from a certain prespecified lag range 1 to m . The result would be a lag structure with varying time lags. Because of our assumptions of a constant delay, this procedure is not further discussed.

In general AC and PAC are a very simple tool for lag selection, but they don't account for complexity like the IC. Therefore they can only be used for a first check if the process is autocorrelated. The PAC may also be used for determination of a maximal lag number m before one of the following lag selection procedures is applied.

3.2.2 The Mutual Information

The mutual information (MI) is - similar to to the AC - a nonparametric measure for the dependence between two series. In the case of time series analysis the two series are the original series x_t and an arbitrary lagged series x_{t-i} . According to Hausser and Strimmer (2009) p.1476 there exists a relationship between the AC^* and the MI. Note that the AC^* in this case means not the Pearson-AC from subsection 3.2.1 but

rater a general version of the AC which also accounts especially for nonlinearity (The Pearson-AC is considered to be only a estimator of the true AC^* in this case).

$$MI_i = MI(x_t, x_{t-i}) = -\frac{1}{2} \log(1 - AC_i^{*2}) \quad (3.2.7)$$

The range of values of the MI is \mathbb{R}^+ . It is symmetric ($MI(x_t, x_{t-i}) = MI(x_{t-i}, x_t)$) and zero if the variable x_t and its lag x_{t-i} are independent (see Hausser and Strimmer (2009) p.1476). Granger and Lin (1994) p. 375 use the relationship between the AC^* and the MI to formulate the MI coefficient (MIC),

$$MIC_i = MIC(x_t, x_{t-i}) = |AC^*| = \sqrt{1 - e^{-2 \cdot MI_i}}. \quad (3.2.8)$$

The MIC of Granger and Lin (1994) is consequently an alternative estimator of the absolute value of the autocorrelation coefficient. Granger and Lin (1994) p.379- 383 show by simulation that their MIC does a better job in identifying the true lag order than the Pearson-AC. The range of the MIC is between 0 and 1, because the MIC approximates an absolute value.

The MI is defined as the joint Shannon-entropy $H(x_t, x_{t-i})$ between the two variables, subtracted the Shannon-entropy of each single variable ($H(x_t)$, $H(x_{t-i})$):

$$MI_i = H(x_t) + H(x_{t-i}) - H(x_t, x_{t-i}) \quad (3.2.9)$$

To explain the Shannon entropy we first need the probability density function of the variable x_t . We will discuss those methods only for $H(x_t)$, but they can as well be applied to calculate the Shannon entropy of x_{t-i} and the Shannon entropy of the joint data x_t and x_{t-i} . Granger and Lin (1994) p.375 propose to use kernel density estimation to determine the probability density function, but in the case of discrete series with a relatively small number of observations the ordinary histogram is sufficient. The histogram is a discrete representation of the distribution. To determine the histogram, the value range of the series is split into d bins. Let $v_i \forall i = 1, \dots, d$ denote the number of values of the original series x_t belonging to the i th bin (for the well known frequency

histogram, the d bins are plotted on the x-axis and the corresponding values v_i on the y-axis). The probability of v_i , $Prob(v_i)$, can then be determined by

$$Prob(v_i) = \frac{v_i}{\sum_{i=1}^d v_i}. \quad (3.2.10)$$

This is called the ML probability estimator (see Hausser and Strimmer (2009) p.1470). Note that

$$\sum_{i=1}^d Prob(v_i) = 1. \quad (3.2.11)$$

The Shannon-Entropy of series x_t is defined as

$$H(x_t) = - \sum_{i=1}^d Prob(v_i) \log(Prob(v_i)). \quad (3.2.12)$$

see Shannon (1948) p.11 and Hausser and Strimmer (2009) p.1470.

In particular if $d \gg \sum_{i=1}^d v_i$ the ML estimator is optimal (Hausser and Strimmer (2009) p.1470). In application such a relation between d and $\sum_{i=1}^d v_i$ is often not observable. Therefore Hausser and Strimmer (2009) pp.1472-1473 propose to use the James-Stein shrinkage estimator, which in this case delivers better results than the ML- and some other estimators. It estimates the probability of v_i by

$$Prob^{SHRINK}(v_i) = \lambda \cdot t_i + (1 - \lambda)Prob(v_i) \quad (3.2.13)$$

whereas usually $t_i = \frac{1}{d}$ and

$$\lambda = \frac{1 - \sum_{i=1}^d (Prob(v_i))^2}{(1 - \sum_{i=1}^d v_i) \sum_{i=1}^d (t_i - Prob(v_i))^2}. \quad (3.2.14)$$

The parameter λ is called the shrinkage intensity. Practical application shows (see section 5.4.1), that the MIC has the same disadvantages as the AC and the PAC for real data series. It possibly tends to indicate no significant lag for some stationary series in a predefined interval $[1; m]$.

3.2.3 Polynomial Approximation Based Lag Selection

Rech, Teräsvirta and Tschernig (2001) propose an alternative nonparametric lag selection procedure based on Taylor polynomials, which are able to approximate unknown nonlinear functions (see section 3.1.1). For our model we assume an AR-NN process with constant delay 1. Before Rech, Teräsvirta and Tschernig (2001)'s method is used, the data have to be stationarized if they are not stationary to avoid spurious regression.

According to the theorem of Stone and Weierstrass, as we have seen in section 3.1.1, the AR-NN can be approximated by a polynomial of sufficient high order. As the procedure of Rech, Teräsvirta and Tschernig (2001) is designed to identify the lag order of an unknown nonlinear function, formula (3.1.2) is used for approximation. This is necessary because it is not yet known if a distribution of the parameters of the AR-NN exists. For lag orders $i = 1, \dots, m$ the nonlinear function is estimated by the polynomial and subsequently evaluated by an IC (the final prediction error (FPE) is used in the original paper, but others are of course possible). Finally the lag order where the model has the smallest IC is chosen.

Rech, Teräsvirta and Tschernig (2001) p.1231 propose to use the OLS method to identify the equation (3.1.2). With the $(T \times m(k))$ variable matrix

$$Z = \underbrace{\begin{pmatrix} X_{1-1} & \dots & X_{1-n} & X_{1-1}X_{1-1} & \dots & X_{1-n}X_{1-n} & \dots & \overbrace{X_{1-n}X_{1-n} \dots X_{1-n}}^{k \text{ times}} \\ X_{2-1} & \dots & X_{2-n} & X_{2-1}X_{2-1} & \dots & X_{2-n}X_{2-n} & \dots & X_{2-n}X_{2-n} \dots X_{2-n} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ X_{T-1} & \dots & X_{T-n} & X_{T-1}X_{T-1} & \dots & X_{T-n}X_{T-n} & \dots & X_{T-n}X_{T-n} \dots X_{T-n} \end{pmatrix}}_{m(k)}$$

and the vectors $X_t = (x_1, x_2, \dots, x_T)^T$, $\Phi = (\phi_{11}, \phi_{12}, \dots, \underbrace{\phi_{TT \dots T}}_{k \text{ times}})^T$ and

$U = (u_1, u_2, \dots, u_T)^T$ equation (3.1.2) can be written as

$$X_t = Z\Phi + U. \quad (3.2.15)$$

An OLS estimator for Φ is

$$\hat{\Phi} = (Z^T Z)^{-1} Z^T X_t. \quad (3.2.16)$$

Therewith a polynomial approximation of the unknown AR-NN function is given by

$$\begin{aligned}
 x_t = & \sum_{j_1=1}^n \phi_{j_1} x_{t-j_1} + \sum_{j_1=1}^n \sum_{j_2=j_1}^n \phi_{j_1, j_2} x_{t-j_1} x_{t-j_2} \\
 & + \sum_{j_1=1}^n \sum_{j_2=j_1}^n \sum_{j_3=j_2}^n \phi_{j_1, j_2, j_3} x_{t-j_1} x_{t-j_2} x_{t-j_3} + \\
 & \sum_{j_1=1}^n \cdots \sum_{j_k=j_{k-1}}^n \phi_{j_1, j_2, \dots, j_k} x_{t-j_1} \cdots x_{t-j_k} + u_t
 \end{aligned}$$

whereas ϕ_{j_1, \dots, j_k} are the elements of $\hat{\Phi}$. Now it is easy to calculate the AIC or BIC for this equation to identify the optimal lag order. Like in linear equations, it is observable in practical application, that the AIC tends to include more lags than the BIC (see section 5.4.1). For application we suggest to use a polynomial of order three, which should be sufficient as it was already proposed by Lee, White and Granger (1993).

3.2.4 The Nonlinear Final Prediction Error

More computational effort is necessary if the nonlinear or nonparametric FPE based on the works of Auestad and Tjøstheim (1990) and Tjøstheim and Auestad (1994) is used. Their work was later extended for lag selection in the presence of heteroscedasticity by Tschernig and Yang (2000). In the following we discuss the version of Tschernig and Yang (2000), which is also mentioned by Medeiros, Teräsvirta and Rech (2006) p.52 as an alternative for lag selection for neural networks. The necessary assumptions in Tschernig and Yang (2000) pp. 3-4 can be subsumed as follows: The function determining the predictable part of the process has to be stationary and differentiable. Further more the process should have a continuously differentiable density. Here the unknown nonlinear function is estimated by a nonparametric general approach, kernel regression. In the original paper nonparametric FPEs are derived based on the Nadaraya-Watson and the local- linear kernel estimator. According to the authors the local linear estimator performs better for nonlinear processes, see Tschernig and Yang (2000) p.13. Hence we concentrate on the local- linear estimator and refer to the original paper for the other possibility.

Instead of specifying the estimator for the function we want to approximate it using a nonparametric function. Let $k(x)$, $k : \mathbb{R} \rightarrow \mathbb{R}$, be the kernel function. It should be positive and symmetric with the property $\int k(x)dx = 1$. Most common is the Gaussian kernel

$$k\left(\frac{x_t - x_{t-i}}{h}\right) = \frac{1}{\sqrt{2\pi}} \cdot e^{-\frac{1}{2} \left(\frac{x_t - x_{t-i}}{h}\right)^2}. \quad (3.2.17)$$

It resembles the Gaussian density function, whereas h denotes the so called bandwidth, which in some sense can be interpreted as an estimator for the variance. A kernel estimator respective the n dimensional vector input X_{t-1} is the product kernel

$$K(X_{t-1}, h) = \frac{1}{h^n} \prod_{i=1}^n k\left(\frac{x_t - x_{t-i}}{h}\right). \quad (3.2.18)$$

For the definition of the local linear estimator for $F(X_{t-1})$ we need the following vectors and matrices:

$$Z = \begin{pmatrix} 1 & 1 & \dots & 1 \\ X_{n-1} - X_{t-1} & X_{n-2} - X_{t-1} & \dots & X_{n-T} - X_{t-1} \end{pmatrix}^T \quad (3.2.19)$$

$$W = \text{diag} \left\{ \frac{K(X_{j-1} - X_{t-1}, h)}{T - n + 1} \right\}_{j=n}^T \quad (3.2.20)$$

$$E = (1, 0, \dots, 0)^T \quad (3.2.21)$$

$$Y = (x_n, \dots, x_T)^T \quad (3.2.22)$$

The dimensions are $((n+1) \times 1)$ for E , $((T-n+1) \times (n+1))$ for Z , $((T-n+1) \times (T-n+1))$ for W and $((T-n+1) \times 1)$ for Y . The matrix in (3.2.20) denotes a diagonal matrix with a diagonal of length $(T-n+1)$ with $\left(\frac{K(X_{j-1} - X_{t-1}, h)}{T-n+1}\right)$ at the j th element for $j = n \dots T$. The local linear estimator for the unknown function $F(X_{t-1})$ is

$$\hat{F}(X_{t-1}, h) = E^T \{Z^T W Z\}^{-1} Z^T W Y. \quad (3.2.23)$$

The nonparametric version of the FPE, called Asymptotic FPE (AFPE) is calculated by

$$AFPE(X_{t-1}, h_{opt}, h_B) = A(h_{opt}) + 2K(0)^n \frac{1}{(T-n+1)h_{opt}^n} \cdot B(h_B), \quad (3.2.24)$$

whereas

$$A(h_{opt}) = \frac{1}{T-n+1} \sum_{i=n}^T \left(x_t - \hat{F}(X_{t-1}, h_{opt}) \right) w(X_{m-t}) \quad (3.2.25)$$

and

$$B(h_B) = \frac{1}{T-n+1} \sum_{i=n}^T \left(x_t - \hat{F}(X_{t-1}, h_B) \right) \frac{w(X_{m-t})}{\mu(X_{t-1}, h_B)}. \quad (3.2.26)$$

The function $w(X_{m-t})$ depends of the prespecified maximal lag m and $\mu(X_{t-1}, h_B)$ is a kernel density estimator calculated by

$$\mu(X_{t-1}, h_B) = \frac{1}{T - n + 2} \sum_{i=n}^{T+1} K(X_{i-1} - X_{t-1}, h_B). \quad (3.2.27)$$

The first term of equation (3.2.24) can be considered as the performance term whereas the second term is the penalty term (Härdle, Kleinow and Tschernig (2001) p.4). For calculation of the plug-in bandwidth h_{opt} used in (3.2.25) we refer to Tschernig (2005) p.10. The bandwidth h_B is determined by the rule of Silverman (1986) pp.86-87, see also Tschernig (2005) pp.9-10:

$$h_B = 2\hat{\sigma} \left(\frac{4}{n+4} \right)^{\frac{1}{n+6}} T^{-\frac{1}{n+6}} \quad (3.2.28)$$

$\hat{\sigma}$ is the geometric mean of the standard deviations of the regressors,

$$\hat{\sigma} = \left(\prod_{i=1}^n \sqrt{\sigma_{x_{t-i}}^2} \right)^{\frac{1}{n}}, \quad (3.2.29)$$

whereas $\sigma_{x_{t-i}}^2$ is the variance of the lag (or regressor) x_i . The lag selection procedure is executed as follows: First the maximal lag order m has to be prespecified and subsequently the AFPEs for all lag orders $n = 1, \dots, m$ have to be calculated. The optimal lag order has the smallest AFPE.

Tschernig and Yang (2000) p.9 mention, that the AFPE tends to include too many lags (overfitting). Thus they propose an extension of the AFPE, the corrected AFPE (CAFPE), which accounts for that problem:

$$CAFPE = AFPE \left(1 + n(T - n)^{-\frac{4}{m+4}} \right) \quad (3.2.30)$$

Like the AFPE the CAFPE has to be minimized. Medeiros, Teräsvirta and Rech (2006) p.52 state that computation of the AFPE or CAFPE may become very effortly, in particular if T and m become large. Thus this procedure may be appropriate especially for small maximal lag numbers m . Already Auestad and Tjøstheim (1990) p.686 observe by simulation, that their version of the nonlinear FPE works only accurate for 3-4 lags in a time series with $T = 500$ observations.

3.3 Parameter Estimation

The most important step to concretize the AR-NN is the estimation of the weights. This equals the estimation of the parameters in linear time series analysis and is called learning or training in neural network theory. Many procedures exist to estimate the parameters of neural networks. Thus it has to be distinguished between supervised learning methods and unsupervised methods (Haykin (2009) pp.64-67). Supervised methods means that the estimation output is compared to a desired output and estimation takes the error signal into account. The error signal is defined as the difference between estimation and desired output. Unsupervised methods use no criteria to control the learning process, so they seem not applicable to statistics and especially time series analysis. Hence in the following we concentrate on supervised learning procedures only.

In general it can be said that there are two different classes of supervised learning procedures concerning the estimation of the parameters (see Widmann (2000) pp.28-29, Haykin (2009) pp.157-159). Batch learning is an iterative procedure where the weights are adjusted in each iteration after the presentation of all T inputs, while during on-line learning - sometimes referred to as stochastic learning - the weights are adjusted on element-by-element basis. This means that for each set of input and output neurons from 1 to T the weights are newly adjusted. The AR-NN process is estimated for the inputs and outputs at a certain time t only, for time $t + 1$ the weights are adjusted again. The main advantages of on-line over batch learning methods are the lower computational complexity and the better adaptability to integrate new values if data arrive sequentially.

Some studies have shown that under certain conditions batch and on-line learning procedures deliver similar results particularly if the input data set is large (Oppner (1998) p.363). In general it can be said that on-line learning is faster and less complex than batch learning, but concerning the precision of the results it performs poor (Bottou (2003) p.172). Hence on-line learning might be more useful in electrical engineering where complexity matters much more than in statistics and particularly time series analysis, where the focus is on comprising long term patterns the most precise way out of a closed historical data set.

Hence in the following we proceed with batch learning procedures only, notwithstanding that in the last years several proceedings have been made concerning the performance of on-line learning procedures in pattern recognition (for example Schraudolph (2002) etc.). We do so because in statistics generally the data sets are completely delivered and contains all the necessary information to analyze the long term interdependencies. On-line learning procedures would be useful if input data were provided continuously, even during the learning process, and therefore the adjustment of the weights has to be an evolving process. A main problem is that on-line algorithms do not really converge, because they are adjusted after each new input of a variable set. The most advantageous property of batch learning is its accurate estimation of the gradient vector for a finite input data set, which guarantees the convergence of the algorithms presented in the following. Hence according to White (1989c) p.1005 batch learning is more efficient than on-line learning from a statistical point of view.

The supervised batch learning algorithms in the subsequent sections are iterative numeric procedures, which possibly lead only to local minima. Direct methods similar to the OLS method can not be implemented for AR-NN as they are not able to account for the nonlinear part of the function. However Medeiros, Teräsvirta and Rech (2006) pp.53-55 propose a hybrid method where the linear part of an AR-NN is estimated using the maximum-likelihood method for linear models. A numeric method estimates the remaining nonlinear part. For example Widmann (2000) pp.65-66 discusses some procedures (simulated annealing and generic algorithms) which claim to be able to identify a global minimum for nonlinear functions. But such procedures do not only require too much effort to be of any use for our purposes (Anders (1997) p.36) but it is also questionable if they can keep their promise and lead to a global minimum at least in finite time (White (1992) pp.111-112, Widmann (2000) p.34). Thus we only concentrate on the local search methods, which are well established in the neural network literature.

3.3.1 The Performance Function

The performance function we use is rooted in White (1989b) pp.430- 433 (see also Trapletti, Leisch and Hornik (2000) pp.2437-2442 and Widmann (2000) pp.32- 34).

Like in the well known least square procedures the goodness of fit of an AR-NN model can be determined by

$$Q(\Theta) = \frac{1}{2} \sum_{t=1}^T (x_t - G(\Theta, X_{t-1}))^2. \quad (3.3.1)$$

If this performance function is used, the parameter estimation procedures in the following are referred to as nonlinear least squares (NLS) method in literature and are not restricted to neural networks only. Of course it is possible to use other performance functions like the likelihood function (see Anders (1997) pp.23-25), but they are less common. As the AR-NN function should also be valid for future values of the time series, the expectation of fuction (3.3.1) has to be minimized. As an AR in general is a stochastic process, there is some uncertainty as we know from definition 2.1. We assume that the uncertainty is only determined by the stochastic part ε_t . This uncertainty should be minimized. Due to our assumptions on ε_t (i.i.d. Gaussian distributed with zero mean and constant variance) this is the case if its variance is minimized. The relationship between the performance function and the variance of ε_t as well as the fact that minimization of the one equals the minimization of the other is formally shown in the following.

The uncertainty also causes the stochastic property of the expectation, which is the reason that one can not calculate Θ directly but has to estimate it. An optimal nonlinear least squares estimator for Θ , $\hat{\Theta}$, can be found by solution of the problem

$$\hat{\Theta} = \arg \min_{\Theta \in \Theta} E(Q(\Theta)), \quad (3.3.2)$$

Θ denotes the network weight space.

By transformation of $Q(\Theta)$ one can show that an optimal $\hat{\Theta} \approx \Theta$ is minimizing the ex-

pected error between an unknown function $F(X_{t-1})$ and its approximation $G(\Theta, X_{t-1})$ (White (1988) p.432):

$$\begin{aligned}
E(Q(\Theta)) &= \frac{1}{2} E\left[\sum_{t=1}^T (x_t - G(\Theta, X_{t-1}))^2\right] \\
&= \frac{1}{2} E\left[\sum_{t=1}^T ((x_t - F(X_{t-1}) + F(X_{t-1})) - G(\Theta, X_{t-1}))^2\right] \\
&= \frac{1}{2} E\left[\sum_{t=1}^T (x_t - F(X_{t-1}))^2\right] + \\
&\quad \frac{1}{2} 2E\left[\sum_{t=1}^T (x_t - F(X_{t-1}))(F(X_{t-1}) - G(\Theta, X_{t-1}))\right] \\
&\quad + \frac{1}{2} E\left[\sum_{t=1}^T ((F(X_{t-1}) - G(\Theta, X_{t-1}))^2)\right] \tag{3.3.3}
\end{aligned}$$

$$\begin{aligned}
&= \frac{1}{2} E\left[\sum_{t=1}^T ((x_t - F(X_{t-1}))^2)\right] + \\
&\quad \frac{1}{2} E\left[\sum_{t=1}^T ((F(X_{t-1}) - G(\Theta, X_{t-1}))^2)\right] \tag{3.3.4}
\end{aligned}$$

$$= \frac{1}{2} E\left[\sum_{t=1}^T \varepsilon_t^2\right] + \frac{1}{2} E\left[\sum_{t=1}^T ((F(X_{t-1}) - G(\Theta, X_{t-1}))^2)\right]. \tag{3.3.5}$$

(3.3.4) follows from (3.3.3) because

$$\begin{aligned}
&E\left[\sum_{t=1}^T ((x_t - F(X_{t-1}))(F(X_{t-1}) - G(\Theta, X_{t-1})))\right] \\
&= E\left[\sum_{t=1}^T ((x_t - F(X_{t-1}))\varepsilon_t)\right] \\
&= E\left[\sum_{t=1}^T (E(x_t - F(X_{t-1}))\varepsilon_t | X_{t-1})\right] \\
&= E\left[\sum_{t=1}^T ((x_t - F(X_{t-1}))E(\varepsilon_t | X_{t-1}))\right] \\
&= 0
\end{aligned}$$

Note that $E(\varepsilon_t | X_{t-1}) = 0 \forall t$. The first term of (3.3.5) states that Θ minimizes the errors of the stochastic part. The second term states that a for $Q(\Theta)$ a minimum is

reached if $G(\Theta, X_{t-1}) = F(X_{t-1})$. In this case (3.3.5) reduces to (with respect to the i.i.d. assumption)

$$Q(\Theta) = \frac{1}{2} \sum_{t=1}^T \varepsilon_t^2. \quad (3.3.6)$$

This causes on the other hand the statement (see therefore equation (3.3.1))

$$\sum_{t=1}^T \varepsilon_t = \sum_{t=1}^T (x_t - G(\Theta, X_{t-1})) \quad (3.3.7)$$

and

$$\varepsilon_t = \varepsilon_t(\Theta) = (x_t - G(\Theta, X_{t-1})). \quad (3.3.8)$$

In equation (3.3.8) the residual at time t is described by a function of Θ . This function as well as the performance function are important for the following sections.

3.3.2 Important Matrix Terms

In the following sections we will extensively use the constructs gradient vector, Jacobian matrix and Hessian matrix. Hence they should be introduced and explained here. The aim is to impart understanding of the dimensionality in the following.

The simplest construct is the gradient vector, indicated by $\nabla(\cdot)$. The gradient vector is a column vector of the partial derivatives of a function respective its variables. Consider for example the gradient vector of the performance function:

$$\nabla Q(\Theta) = \begin{pmatrix} \frac{\partial Q(\Theta)}{\partial \theta_1} \\ \vdots \\ \frac{\partial Q(\Theta)}{\partial \theta_r} \end{pmatrix} \quad (3.3.9)$$

Its dimension is $(r \times 1)$. If a function additionally depends on time t , for example the residual at time t in equation (3.3.8), the Jacobian matrix corresponds in some sense to the gradient vector. It is the matrix of the partial derivatives respective the variables

(in the rows) and the time (in the columns). We denote it by $J(\cdot)$. For equation (3.3.8) the Jacobian matrix has dimension $(T \times r)$ and is calculated by:

$$J(\varepsilon_t(\Theta)) = \begin{pmatrix} \frac{\partial \varepsilon_1}{\partial \theta_1} & \frac{\partial \varepsilon_1}{\partial \theta_2} & \cdots & \frac{\partial \varepsilon_1}{\partial \theta_r} \\ \frac{\partial \varepsilon_2}{\partial \theta_1} & \frac{\partial \varepsilon_2}{\partial \theta_2} & \cdots & \frac{\partial \varepsilon_2}{\partial \theta_r} \\ \vdots & & \ddots & \vdots \\ \frac{\partial \varepsilon_T}{\partial \theta_1} & \frac{\partial \varepsilon_T}{\partial \theta_2} & \cdots & \frac{\partial \varepsilon_T}{\partial \theta_r} \end{pmatrix}. \quad (3.3.10)$$

There is a relationship between the gradient vector of the performance function and the Jacobian matrix of (3.3.8) which can be constructed using the residual vector $E = (\varepsilon_1, \varepsilon_2, \dots, \varepsilon_T)$. The $(r \times 1)$ -dimensional gradient vector $\nabla Q(\Theta)$ is the product of the transposed Jacobian matrix $J(\varepsilon_t(\Theta))$ of dimension $(r \times T)$ and the $(T \times 1)$ -dimensional vector E .

$$\nabla Q(\Theta) = J(\varepsilon_t(\Theta))^T E = \begin{pmatrix} \varepsilon_1 \frac{\partial \varepsilon_1}{\partial \theta_1} + \varepsilon_2 \frac{\partial \varepsilon_2}{\partial \theta_1} + \cdots + \varepsilon_T \frac{\partial \varepsilon_T}{\partial \theta_1} \\ \varepsilon_1 \frac{\partial \varepsilon_1}{\partial \theta_2} + \varepsilon_2 \frac{\partial \varepsilon_2}{\partial \theta_2} + \cdots + \varepsilon_T \frac{\partial \varepsilon_T}{\partial \theta_2} \\ \vdots \\ \varepsilon_1 \frac{\partial \varepsilon_1}{\partial \theta_r} + \varepsilon_2 \frac{\partial \varepsilon_2}{\partial \theta_r} + \cdots + \varepsilon_T \frac{\partial \varepsilon_T}{\partial \theta_r} \end{pmatrix} \quad (3.3.11)$$

Note that this relationship is in particular $\forall i = 1, \dots, r$ based on (3.3.6) because

$$\frac{\partial Q(\Theta)}{\partial \theta_i} = \nabla \frac{1}{2} \left(\sum_{t=1}^T \varepsilon_t^2 \right) = 2 \cdot \frac{1}{2} \left(\sum_{t=1}^T \varepsilon_t \right) \frac{\partial \left(\sum_{t=1}^T \varepsilon_t \right)}{\partial \theta_i}. \quad (3.3.12)$$

The third important matrix term is the Hessian matrix. It is the matrix of second order derivatives of a function respective its input variables and is denoted by $\nabla^2(\cdot)$. For the performance function $Q(\Theta)$ it has the following $(r \times r)$ representation:

$$\nabla^2 Q(\Theta) = \begin{pmatrix} \frac{\partial^2 Q(\Theta)}{\partial \theta_1 \partial \theta_1} & \frac{\partial^2 Q(\Theta)}{\partial \theta_1 \partial \theta_2} & \cdots & \frac{\partial^2 Q(\Theta)}{\partial \theta_1 \partial \theta_r} \\ \frac{\partial^2 Q(\Theta)}{\partial \theta_2 \partial \theta_1} & \frac{\partial^2 Q(\Theta)}{\partial \theta_2 \partial \theta_2} & \cdots & \frac{\partial^2 Q(\Theta)}{\partial \theta_2 \partial \theta_r} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 Q(\Theta)}{\partial \theta_r \partial \theta_1} & \frac{\partial^2 Q(\Theta)}{\partial \theta_r \partial \theta_2} & \cdots & \frac{\partial^2 Q(\Theta)}{\partial \theta_r \partial \theta_r} \end{pmatrix} \quad (3.3.13)$$

3.3.3 Basic Features of the Algorithms

All numeric parameter estimation algorithms for neural networks work the same way: Starting with a random initial parameter vector Θ^0 it is searched iteratively for the optimal parameter vector. The optimal parameter vector is reached if the performance function is minimized. To show how the search direction is determined the performance function is depicted in figure 3.3 as a function of Θ^i , whereas i indicates the number of the iteration, starting with $i=0$. The function has various extrema, minima as well as maxima, which satisfy $\nabla Q(\Theta) = 0$ (Bishop (1995) pp.254-255), because the gradient $\nabla Q(\Theta)$ is the slope of the function.

However sometimes only a local minimum might be reached by the algorithm. In addition the choice of the initial weight vector Θ^0 influences the outcome of the algorithm respective local and global minima. But as already discussed above, there are no alternative algorithms which guarant to find a global minimum.

In general the algorithm is carried out according to the flow chart in figure 3.4. The weights are updated after each iteration and the performance function is calculated. If a stopping criterion is reached, the algorithm is quitted. The stopping criterion can be a restriction concerning the performance function, for example the distance between performance function in two iterations, which should be below a certain value (see Anders (1997) p.36). For our attempt such a stopping criterion eventually circumvents

the detection of a better minimum, because the algorithm is stopped immediately after the criterion is achieved. Thus we recommend saving the result of the performance function and the parameter vector after each iteration. Then the maximal number of iterations i^{max} can be used as a stopping criterion and the optimal parameter vector is calculated by the following steps:

- Start the algorithm with the initial weight vector Θ^0
- After each iteration save $Q(\Theta^i)$ and Θ^i
- Quit the algorithm after i^{max} iterations
- Among the saved values search

$$Q(\Theta^{i^*}) = \arg \min_{i \in [0; i^{max}]} Q(\Theta^i)$$

- Θ^{i^*} is the optimal parameter vector

i^* denotes the iteration where the optimal parameter vector is reached. Such a procedure can be interpreted as search for a global minimum within a finite horizon of iterations. One critical point is the need for storage for each iteration, but to our attempt this is outweighed by the fact that this procedure is able to identify a very good local minimum. Often the performance function converges to a certain constant within a finite number of iterations. Therefore a good local minimum within a finite number of iterations, i^{max} , often is in fact a global minimum. Thus we prefer this method in the following. It is implemented in context with the Levenberg-Marquardt algorithm in R in appendix B.4.

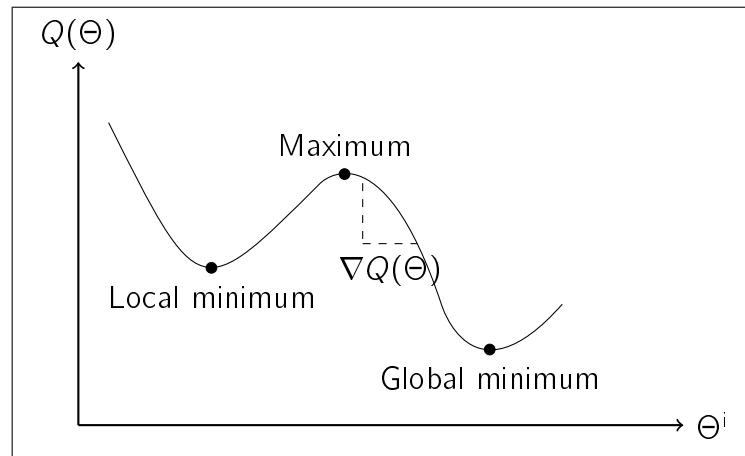


Figure 3.3: Iterative parameter estimation

Source: Authors' design, based on Bishop (1995) p.255 figure 7.2

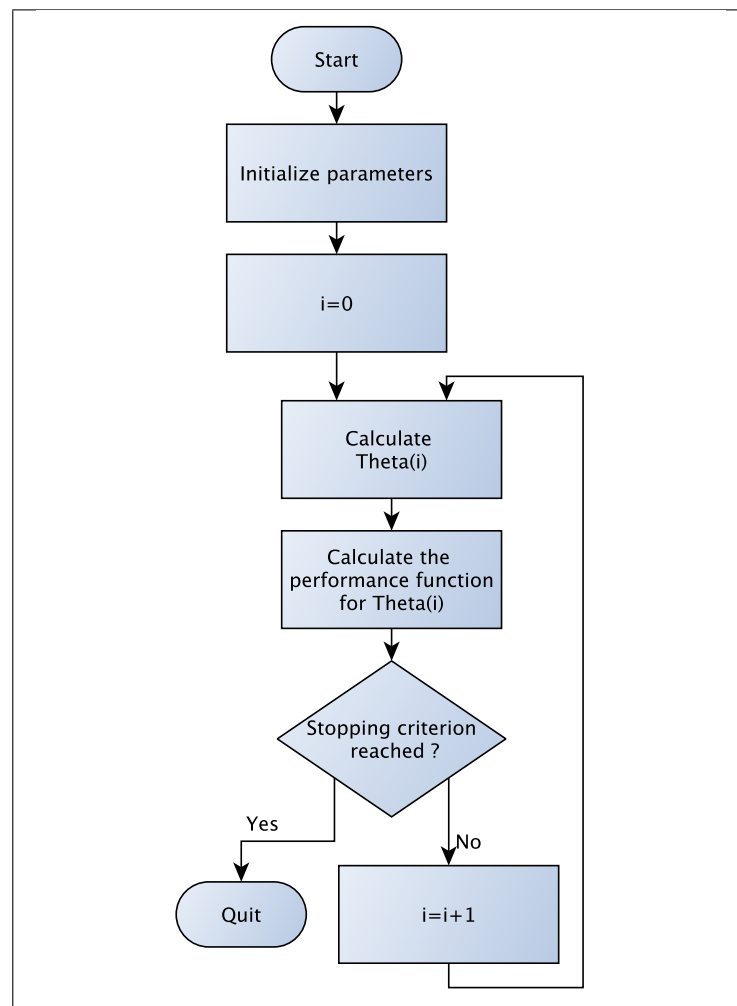


Figure 3.4: Flow chart iterative parameter estimation

Source: Authors' design, based on the figures in Anders (1997) p.37 and pp.127-132

3.3.4 First Order Gradient Descent Methods

In this section the oldest and computationally simplest iterative parameter estimation algorithms for neural networks are discussed. They are based on the first order partial derivatives - the gradient vector - of the performance function, $\nabla Q(\Theta)$. Therefore they are called first order gradient descent methods. Rumelhardt, Hinton and Williams (1986a) p. 535 propose to compute the changes of the weights proportional to the accumulated partial derivatives. This learning algorithm is also called steepest descent algorithm (see Bishop (1995) p. 263). The change in the individual weight $\theta_i \in \Theta$ is

$$\Delta\theta_i^{i+1} = -\nu \frac{\partial Q(\Theta^i)}{\partial \theta_i} \quad (3.3.14)$$

whereas the parameter $\nu \in \mathbb{R}^+$ is called the learning rate, $i = 1, \dots, r$ and i is the number of the iteration, see Rumelhart, Hinton and Williams (1986b) p.323. Starting with an arbitrary initial Θ_0 , the weights are updated after each iteration. Equation (3.3.14) can be written in vector representation (see Widmann (2000) p.40):

$$\Delta\Theta^{i+1} = -\nu \nabla Q(\Theta^i), \quad (3.3.15)$$

$\nabla Q(\Theta^i)$ is the gradient vector of dimension $(r \times 1)$. The main problem with the steepest descent algorithm is the choice of an appropriate learning rate. If it is chosen too small, many steps are necessary, because the changes after each iteration are very small. If in contrast the learning rate is chosen too big, the danger consists to overlook a global minimum, because the results may tend to strong oscillation. Varying the learning rate is a subjective and therefore not recommendable solution (Bishop (1995) pp.264-266).⁵

Several extensions of the steepest descent algorithm have therefore been developed to systematize the method. The first to mention here, also proposed in Rumelhart, Hinton and Williams (1986b) p.330, is to include a momentum term in (3.3.15)

$$\Delta\Theta^i = -\nu \nabla Q(\Theta^{i-1}) + \alpha \Delta\Theta^{i-1}, \quad (3.3.16)$$

⁵Note that this simple version of the first order gradient descent method is often called backpropagation algorithm. This is not correct, as backpropagation only describes the way derivatives are calculated. The backpropagation method was also published in the paper of Rumelhardt, Hinton and Williams (1986a). As we use only three layers, the derivatives can be reconstructed using the chain rule of differentiation. For higher order layer networks certainly a detailed discussion of the backpropagation method, see for example Bishop (1995) pp. 140-146, would be necessary.

where $\alpha \in [0; 1]$ is the momentum parameter and the last term is called the momentum term. The reason why this term is added is that it can filter out high frequency variations in the error-surface in the weight space. In other words the momentum term "smoothens" the oscillations. The effect is faster convergence of the algorithm, because one can use larger ν without the danger of missing any global minimum (Rumelhart, Hinton and Williams (1986b) p.330). According to Bishop (1995) p.268 this extension does not really solve the problems of the simple gradient descent algorithm of equation (3.3.15), because it depends on a second parameter, α , which has to be chosen arbitrary like ν .

Another alternative is the so called bold-driver method of Vogl et al. (1988), where the learning rate is updated according to some rules after each iteration, for the following see Vogl et al. (1988) p.259 and Bishop (1995) p.269. Consider equation (3.3.15): If the value of the present error function $Q(\Theta^i)$ is smaller than that of the previous error function $Q(\Theta^{i-1})$, the learning rate can be slightly increased in the next iteration. This is done by multiplication by a parameter ϕ_1 , which is slightly above 1, for example $\phi_1 = 1.1$. If the value of the present error function is bigger than that of the previous error function, the learning rate has to be decreased and the iteration has to be repeated, because it can be possible that a global minimum has been overlooked. Therefore α is set 0 and ν is multiplied by a parameter ϕ_2 , which has to be significantly less than 1, for example $\phi_2 = 0.5$.

A modification of the bold-driver method is the delta-bar-delta-rule proposed by Jacobs (1988) pp.299-301. A local gradient "delta" is defined for iteration i and weight i as

$$\delta_i^i = \frac{\partial Q(\Theta^i)}{\partial \theta_i^i}. \quad (3.3.17)$$

The extension of that local gradient, "delta-bar", is

$$\bar{\delta}_i^i = (1 - \kappa_1)\delta_i^i + \kappa_1\bar{\delta}_i^{i-1}. \quad (3.3.18)$$

The learning rate update is computed using (3.3.17) and (3.3.18) by

$$\Delta \nu_i^i = \begin{cases} \kappa_2 & \text{if } \bar{\delta}_i^{i-1}\delta_i^i > 0 \\ -\kappa_3\nu_i^i & \text{if } \bar{\delta}_i^{i-1}\delta_i^i < 0 \end{cases}. \quad (3.3.19)$$

Note that this method implies, that each element of the weight vector is updated individually like in the bold-driver algorithm. The system behind this approach is on the one hand to increase the learning rates in a linear way and thus to avoid a too rapid increase. On the other hand the learning rate is decreased exponentially to ensure that no global minimum is missed. One disadvantage of this method is, that three parameters ($\kappa_1, \kappa_2, \kappa_3$) have to be determined, if a momentum term is added four parameters. The other disadvantage is, that the weight parameters are treated like they were independent, which actually is often not the case (Bishop (1995) p.271).

Another class of first order gradient descent methods are the so called line search methods (Bishop (1995) p.272-274). They are at least an extension of the gradient-reuse algorithm of Hush and Salas (1988). The difference to the ordinary steepest descent algorithm is, that the search direction is not determined only by the negative local gradient but also by the weight space. Gradients are “reused” to update the learning rate. Formally this can be written by using

$$\begin{aligned}\Delta\Theta^{i+1} &= -\nu^i \nabla Q(\Theta^i) \\ \Theta^{i+1} &= \Theta^i - \nu^i \nabla Q(\Theta^i),\end{aligned}\tag{3.3.20}$$

whereas the parameter ν^i is the parameter which minimizes

$$Q(\nu^i) = Q(\Theta^i - \nu^i \nabla Q(\Theta^i)).\tag{3.3.21}$$

According to Widmann (2000) p.40 one avoids therewith to go arbitrary far in the direction of the gradient. The learning rate ν^i leads to the deepest point in this direction. There are two practical approaches to find the minimum of (3.3.21): The first is a quadratic interpolation, which in a first stage computes three values of ν , $a < b < c$ such that $Q(a) > Q(b)$ and $Q(c) > Q(b)$. If the error function is continuous, it is guaranteed that a minimum is between a and c . The second stage is the location of the minimum by adjusting a quadratic polynomial to the error function at the points a, b and c and searching for a minimum of that function (Bishop (1995) p.273). The alternative approach is the computation of the derivatives of (3.3.21). This requires much more computational effort and would in fact result in a second order gradient descent method (Widmann (2000) p.41).

Updating the search direction in line-search methods, in other words the gradient, is the idea of the so called conjugate gradient procedures. The reason for those methods is, that sometimes the gradient is not the optimal search direction and therefore high computational effort is necessary to find the minimum of (3.3.21) (Bishop (1995) pp.274-275). r^i is a $(r \times 1)$ -dimensional vector for the search direction at iteration i , such that (3.3.20) can be written as

$$\Theta^{i+1} = \Theta^i - \nu^i r^i. \quad (3.3.22)$$

A solution of the mentioned problem is to choose r^{i+1} such that

$$(r^i)^\top (\nabla^2 Q(\Theta^i)) r^{i-1} = 0. \quad (3.3.23)$$

The search direction is in this case said to be conjugate (Bishop (1995) p.276). The algorithms to calculate the r 's can be described according to Bishop (1995) pp.274-283 as follows: The initial value is calculated by $r^0 = -\nabla Q(\Theta^0)$. The subsequent search directions can be found by

$$r^{i+1} = \nabla Q(\Theta^{i+1} + \alpha^i r^i). \quad (3.3.24)$$

The weight vector for the iterations is determined by (3.3.22), the learning rate by line-search methods. For the determination of the parameter α^i there are three well known methods. It is computed by the formula of Hestenes and Stiefel (1952) by

$$\alpha^i = \frac{(\nabla Q(\Theta^{i+1}))^\top (\nabla Q(\Theta^{i+1}) - \nabla Q(\Theta^i))}{(r^i)^\top (\nabla Q(\Theta^{i+1}) - \nabla Q(\Theta^i))}, \quad (3.3.25)$$

according to the formula of Polak and Ribière (1969) (a modification of the above formula to avoid to include r^i in the formula, see Bishop (1995) p.280) by

$$\alpha^i = \frac{(\nabla Q(\Theta^{i+1}))^\top (\nabla Q(\Theta^{i+1}) - \nabla Q(\Theta^i))}{(\nabla Q(\Theta^i))^\top (\nabla Q(\Theta^i))} \quad (3.3.26)$$

and according to the formula of Fletcher and Reeves (1964) by

$$\alpha^i = \frac{(\nabla Q(\Theta^{i+1}))^\top (\nabla Q(\Theta^{i+1}))}{(\nabla Q(\Theta^i))^\top (\nabla Q(\Theta^i))}. \quad (3.3.27)$$

The conjugate gradient algorithm is designed for quadratic error functions with positive definite Hessian matrix. If it is applied to arbitrary error functions it is assumed that they

can be locally approximated by a quadratic polynomial, but they don't have necessarily to be quadratic. The formula of Polak and Ribière (1969) is superior in the case of nonlinear error functions (Haykin (2009) p.222). But continuing this discussion would lead too far as we only use the quadratic error function (3.3.1). The conjugate-gradient method is generally the most powerful first order gradient descent method (if we abstain from quasi-Newton methods mentioned in the next chapter).

3.3.5 Second Order Gradient Descent Methods

The second order gradient descent methods are learning algorithms, which explicitly make use of the Hessian matrix $\nabla^2 Q(\Theta)$. Consider equation (3.3.14): The learning rate ν is replaced by the inverse Hessian matrix such that

$$\Theta^{i+1} = \Theta^i - (\nabla^2 Q(\Theta^i))^{-1} \nabla Q(\Theta^i). \quad (3.3.28)$$

The second term in this equation is called Newton direction. Its main advantage is, that the Newton direction or Newton step of a quadratic error function directly points towards a minimum and hence avoids oscillation (Bishop (1995) p. 286).

However determining the Hessian matrix brings along some problems. Firstly, it is very demanding from a computational point of view to calculate and invert the Hessian matrix. To show this, let \mathcal{O} denote the Landau symbol indicating the upper bound of the computing complexity. Computing the Hessian matrix has complexity $\mathcal{O}(r^2)$ and inverting it $\mathcal{O}(r^3)$ (Bishop (1995) p.287). Secondly, the Newton direction may point to a maximum or saddle point, which is the case if the Hessian matrix is not positive definite. As a consequence the error is not necessarily reduced in each iteration. Thirdly, the step size of the Newton step may be such large that it leads out of the range of validity.

The second problem can be reduced by adding a positive definite symmetric matrix to the Hessian matrix which includes the unit Matrix \mathbf{I} and a sufficient large parameter λ . Then the combination

$$\nabla^2 Q(\Theta) + \lambda \mathbf{I} \quad (3.3.29)$$

is certainly positive definite. This is a compromise between the negative gradient search direction, which (approximatively) results if λ is chosen large, and the Newton direction,

which results if λ is very small. In this way also the third problem mentioned above is solved. But the first problem remains, which is in general known as the greatest disadvantage of the Newton's method and is the origin of several approximation procedures called quasi-Newton methods. Because they do not deal with second order gradients directly but approximate them via first order gradients, they are in general classified as first order gradient methods (Bishop (1995) pp.287-290, Widmann (2000) pp.44-45, Haykin (2009) pp.224-226). We will not discuss those methods here and refer to the literature mentioned above as with the Levenberg-Marquardt algorithm in the next chapter a powerful kind of a quasi-Newton method is shown.

Shortly we want to compare the quasi-Newton and conjugate gradient methods (see Haykin (2009) p.226-227): The complexity of conjugate gradient is only $\mathcal{O}(r)$, thus this method is preferable to quasi-Newton methods with an overall computing complexity of $\mathcal{O}(r^2)$ (Haykin (2009) p.227) if the weight vector becomes large. An additional argument to that point of view is, that storage is required for the approximated Hessian matrices which of course becomes the larger the more elements are included in the weight vector. In the close neighborhood of a local minimum however, quasi-Newton methods converge faster as they approximate Newton's method accurate. Quasi-Newton, conjugate gradient and the Levenberg-Marquardt algorithm converge with superlinear speed, whereas the other methods converge with linear speed (Bottou (2003) pp. 165-166).

3.3.6 The Levenberg-Marquardt Algorithm

In the following we continue with the Levenberg- Marquardt algorithm (founded in the works of Levenberg (1944) and Marquardt (1963)) which combines the steepest descent algorithm of Rumelhart, Hinton and Williams (1986b) and Newton's method. Like other quasi-Newton methods it can not be counted to the second order gradient methods, because the Hessian matrix is approximated by combinations of the Jacobian matrix of $\varepsilon_t(\Theta)$, the matrix of first order gradients, such that no second order gradients rest to calculate. According to Haykin (2009) p.227 the advantages of this method are therefore that it converges rapidly like Newton's method but it can not diverge because of the steepest descent algorithm influence. Via modification of some parameters the Levenberg-Marquardt algorithm can be made equal to either the steepest descent- or Newton's algorithm. The algorithm is also recommended in several econometric works as for example in Medeiros, Teräsvirta and Rech (2006) p.54 and is commonly

known as one of the most powerful learning methods for neural networks. According to Bishop (1995) p.253 the Levenberg-Marquardt algorithm is especially applicable to error-sum-of-squares performance functions. Therefore it is used for our empirical examples in chapter 5. Appendix B.4 contains a implementation of the algorithm in the statistical programming language R. For the following description see Hagan and Menhaj (1994) p.990:

We can show that the Hessian matrix of the performance function can be estimated by the cross product of the Jacobian matrices of $\varepsilon_t(\Theta)$. The following relationships are in the style of equation (3.3.11):

$$\begin{aligned}\nabla^2 Q(\Theta) &= \nabla^2 \left(\frac{1}{2} \sum_{t=1}^T \varepsilon_t(\Theta)^2 \right) \\ &= \nabla \left(\nabla \left(\frac{1}{2} \sum_{t=1}^T \varepsilon_t(\Theta)^2 \right) \right) \\ &= \nabla (J(\varepsilon_t(\Theta))^T E)\end{aligned}\tag{3.3.30}$$

On element by element basis we get for the i th row of (3.3.30) and the j th weight, $i, j = 1, \dots, r$, by the product rule of differentiation:

$$\frac{\partial \left(\sum_{t=1}^T \varepsilon_t \frac{\partial \varepsilon_t}{\partial \theta_i} \right)}{\partial \theta_j} = \sum_{t=1}^T \left(\frac{\partial \varepsilon_t}{\partial \theta_j} \frac{\partial \varepsilon_t}{\partial \theta_i} + \varepsilon_t \frac{\partial^2 \varepsilon_t}{\partial \theta_i \partial \theta_j} \right)\tag{3.3.31}$$

The second term on the right hand of (3.3.31) is approximatively zero, see Hagan and Menhaj (1994) p.990 and Bishop (1995) p.291. The first term equals the cross product of the Jacobian matrices. With this result we get for (3.3.28)

$$\Delta \Theta^{i+1} = - [J(\varepsilon_t(\Theta^i))^T J(\varepsilon_t(\Theta^i))]^{-1} J(\varepsilon_t(\Theta^i))^T E^i.\tag{3.3.32}$$

The fact that the pure crossproduct of the Jacobian matrices sometimes leads to singularities as application shows might be problematic. Thus an modification is needed. Equation (3.3.28) together with (3.3.29) can be written as

$$\Delta \Theta^{i+1} = - [\nabla^2 Q(\Theta^i) + \lambda \mathbf{I}]^{-1} \nabla Q(\Theta^i).\tag{3.3.33}$$

The Levenberg- Marquardt representation of equation (3.3.33) now contains the approximation (3.3.32) of (3.3.29):

$$\Delta\Theta^{i+1} = - [J(\varepsilon_t(\Theta^i))^T J(\varepsilon_t(\Theta^i)) + \lambda \cdot \mathbf{I}]^{-1} J(\varepsilon_t(\Theta^i))^T E^i \quad (3.3.34)$$

λ is multiplied by a factor τ if an iteration results in an increased $Q(\Theta)$. If an iteration reduces $Q(\Theta)$, λ is divided by τ . If λ and τ are chosen to be such big that their influence is stronger than that of the Hessian matrix, the Levenberg- Marquardt algorithm becomes very similar to the steepest descent algorithm of Rumelhardt, Hinton and Williams (1986a). If those parameters are chosen to be zero, the Levenberg- Marquardt algorithm is identical to the Newton procedure. For computational reasons the parameter λ should at least be different from zero such that the matrix $[J(\varepsilon_t(\Theta^i))^T J(\varepsilon_t(\Theta^i)) + \lambda \cdot \mathbf{I}]$ is positive definite (see section 3.3.5 and Haykin (2009) p.228). The flowchart in figure 3.5 explains how the algorithm runs. Because of the squared Jacobian matrices in (3.3.34) (also referred as Gauss-Newton method) one abstains from calculating the complex Hessian matrices and has consequently all the advantages of the Newton algorithms combined with less complexity.

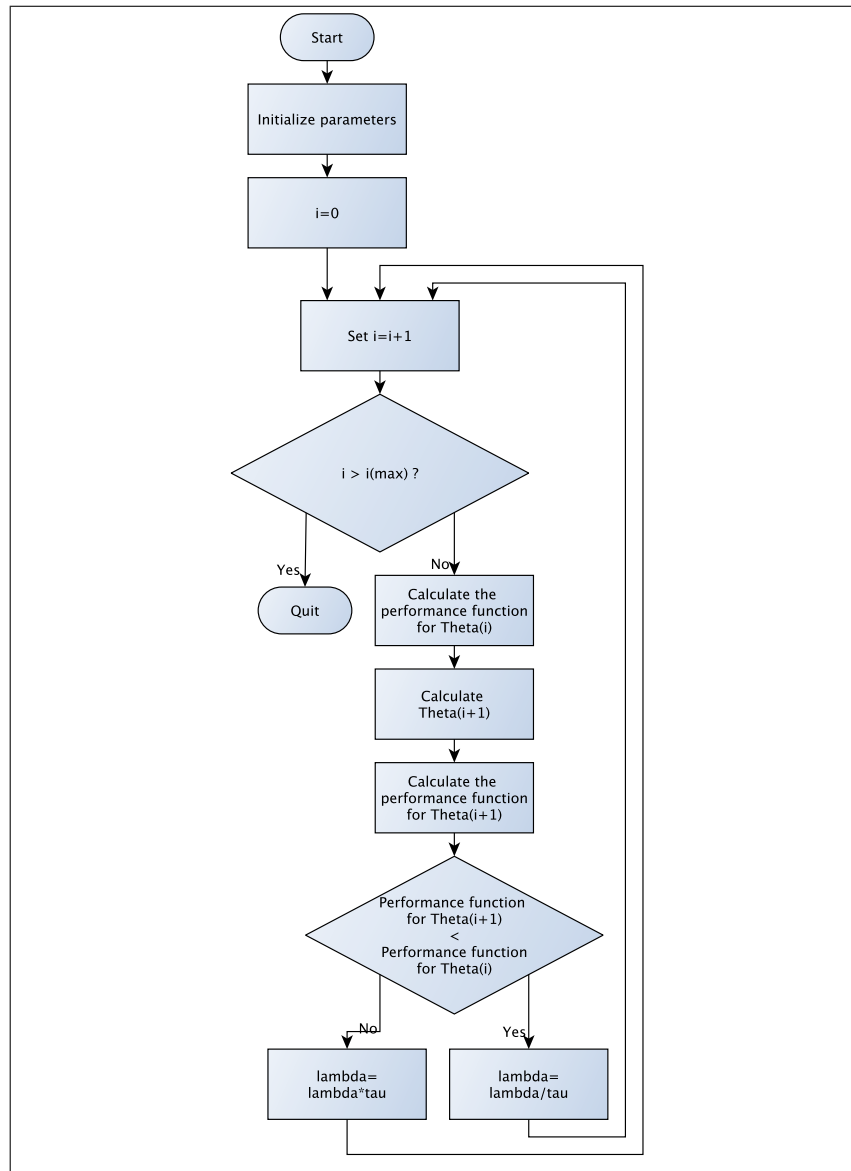


Figure 3.5: Flow chart Levenberg-Marquardt algorithm

Source: Authors' design, based on the figures in Anders (1997) p.37 and pp.127-132

3.3.7 Stopped Training

Using one of the learning algorithms discussed above, it is possible - at least in theory - to minimize the stochastic part such that it vanishes if $i \rightarrow \infty$. This leads to the assumption that the AR-NN becomes a perfect estimator for the process. To consider to which problems this may eventually lead, we have to mention the aims of the analysis of the process. We want to detect the long run behavior of the process, which is assumed to be included in the considered data series. Having discovered this behavior once, values over a certain period in the future can be forecasted, showing a trend which is only disturbed by the stochastic part ε_t . Its influence is only temporary. A neural network with (nearby) zero residual variance would put in question our general model of the process (equation (2.1.1)) because a deterministic model without stochastic part might then be a better representation. According to Medeiros, Teräsvirta and Rech (2006) p.51 this question arises only with nonlinear models. Concerning out-of-sample prediction, AR-NN's with low residual variance sometimes behave surprisingly poor. This phenomenon is called overfitting (or sometimes overlearning, see Haykin (2009) p.194). The overfitted network has lost its ability to generalize (Haykin (2009) p.194). According to Widmann (2000) p.56-57, overfitted neural networks are too big respective the free parameters compared to the necessary complexity for the analysis of the data. Thus the model is adjusted to the data the more precisely the more weights are involved, but its property to explain the underlying process is not changed or even gets lost if it is overfitted.

To visualize overfitting, we use an example with real data. Figure 3.6 shows an overfitted AR-NN, generated with data from chapter 5 (differences of the log(USD/EUR) exchange rate, 128 values) and the Levenberg-Marquardt function in appendix B.4.⁶ For estimation an AR-NN with 20 hidden neurons was used. The data set is splitted into two subsets: The first 120 values are used for estimation, the last 8 values for comparison to a 8 step forecast. We observe that the model fits very well in-sample, whereas the forecast predicts values which are not at all realistic. They are even not in the range of the original values. Such observations lead to two conclusions: Firstly the model of the process is wrong. An overtrained network will not only be adjusted to the deterministic part, but also to the stochastic part (Widmann (2000) p.56). Hence the network is no longer the concretization of equation (2.1.1) and is not able to provide

⁶The function has been modified for generating this figure

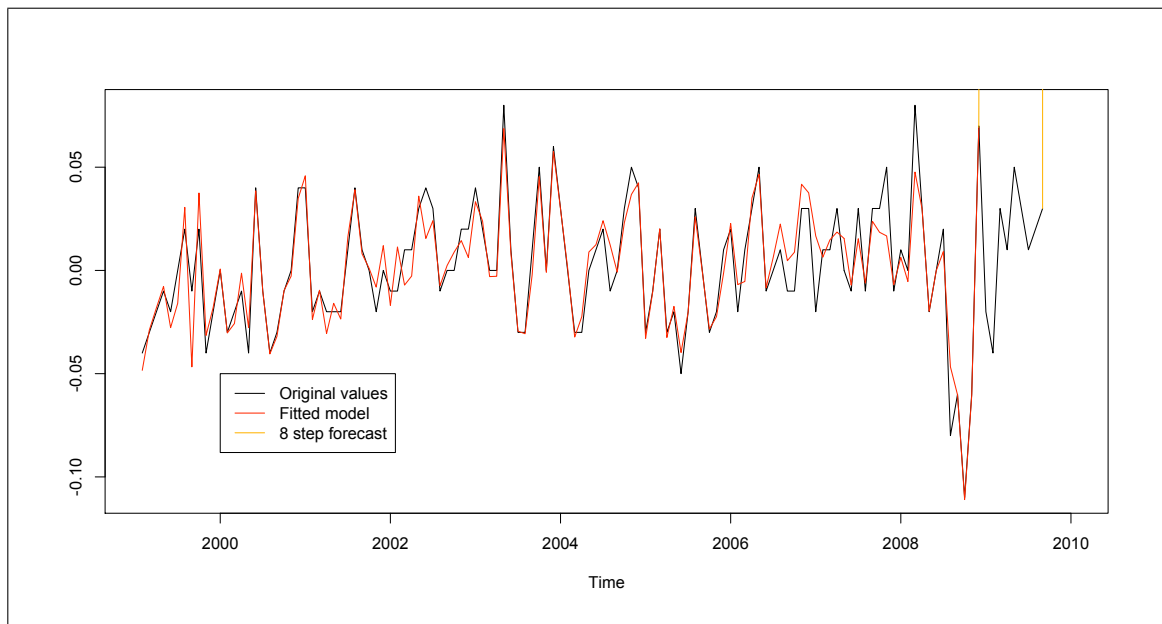


Figure 3.6: Example: Overfitted AR-NN

Source: Authors' design

the expectation of this equation. The stochastic part is therefore an essential part of the equation, which describes the dynamics of the process. The second conclusion is: We need some stopping conditions which prevent the learning algorithm from adjusting to the stochastic part.

A well known approach to solve the problem of overfitting is stopped training or early stopping. The data set used for estimation is partitioned in an estimation subset (ES) and a validation subset (VS). According to Haykin (2009) pp.202-203 in general an estimation subset containing 80% of the values and a validation subset containing the remaining 20% is a good partition. For alternative ways on how to partition the data see the referred literature. If the model should be used for prediction, empirical application shows that the validation subset should be as large as the number of steps to predict (for example if the model is used for a 8 step forecast, the validation subset should contain about 8 values). In our empirical part we observe that the AR-NN perform very well in the short run (1-step or 2-step) forecasts. Consequently the validation subset should be correspondingly small.

According to the stopped training method after each iteration the resulting parameters are used to forecast the values of the validation subset. Subsequently the sum of

squares of the deviations of the forecasted from the original validation subset values is calculated (in the following referred as VS-RSS). It is observable that the VS-RSS is only minimized up to a finite optimal number of iterations i^* . If more iterations are used, the VS-RSS will increase. The ES-RSS in contrast always decreases with increasing number of iterations. Figure 3.7 shows a sketch how ES-RSS and VS-RSS develop during the iterations.

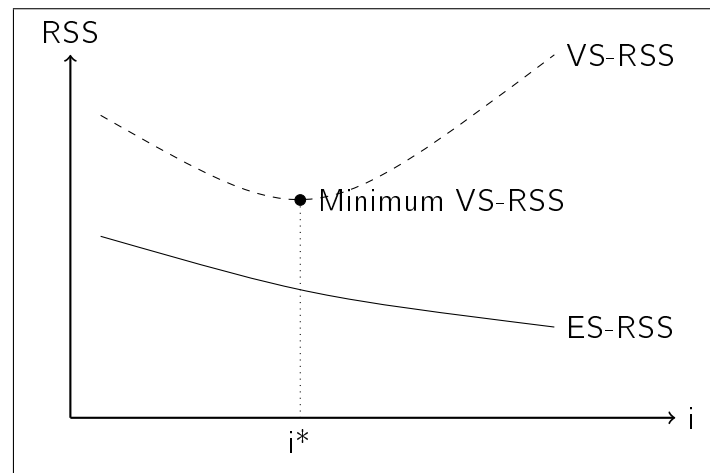


Figure 3.7: Stopped training: Development of ES-RSS and VS-RSS during the learning algorithm

Source: Authors' design, based on Bishop (1995) p.364 figure 9.16

The parameter vector at the iteration where the VS-RSS is minimal (i^*) is considered to be the optimal parameter vector. Empirical application shows that an AR-NN with a parameter vector chosen by the stopped training method is able to produce forecasts which are at least in the range of the original values.

In application it is sometimes observable, that the results of the Levenberg-Marquart algorithm depend on the initial values for the parameter vector. Stopped training provides a simple way to overcome this problem: The search for the minimum VS-RSS does not start until some initial iterations are executed (in the empirical part for example five initial iterations are used). In this way the initial values are modified by the first iterations in order to be more appropriate. In particular empirical application for this dissertation showed that for such a procedure the difference in the results for various initial values is lowered. Another fact that has been observed was that solutions at very low numbers of iterations (iterations < 5) often do not perform very well in out-of-sample prediction.

Stopped training is just a method of limiting the iterative parameter estimation process and not yet a solution for the problem of potentially too many hidden neurons. The general approach to that problem is finding the simplest model from a set of models with the same goodness of fit the simplest model (Bishop (1995) p.14). Similar to linear time series analysis it is possible to check wheater the influence of certain parameters is significant for the model. Further if only the first $(1 + n)$ parameters were sufficient for the model, a linear model would be a better approach. Therefore such tests are sometimes referred to as linearity tests, see section 3.1. Of course there are some ex-ante parameter determination methods (for example the rule Baum and Haussler (1988)), but they are not so powerful as the ex-post parameter tests for they are not built on a sound theoretical fundament like parameter hypothesis tests. Therefore we abstain from explaining them (for an overview see Widmann (2000) pp.57-65) and recommend the testing procedures discussed in the following sections.

3.4 Parameter Tests

The final step is now to examine if the estimated model is appropriate. This can be performed in two ways:⁷ Bottom-up means starting with the estimated model. It is examined, if an additional hidden unit would improve the model. Therefore the non-linearity test already known from section 3.1 is extended. Their disadvantage is, that they only consider the in-sample performance (for example the in-sample RSS) of the models. An alternative is cross validation, which means that the quality of a model is evaluated by its out-of-sample performance.

Top-down parameter tests are well known from linear statistics. They consider one parameter or a set of parameters within the estimated model and test them on significance. For an estimated model the general procedure is first to execute bottom-up parameter tests as long as no additional hidden unit improves the model. Then a model including those additional hidden units is again estimated and evaluated by top-down parameter tests (see figure 3.1 for an overview how top-down and bottom-up parameter tests are used).

⁷The notations bottom-up and top-down parameter testing are taken from Anders (1997) pp.127-128

3.4.1 Bottom-Up Parameter Tests

In this section two methods for bottom-up evaluation of an estimated model are discussed. The first method is based on Taylor polynomial estimation of an additional hidden neuron. Therefore the test of Lee, White and Granger (1993) (see also section 3.1.2.2) is slightly modified by Teräsvirta, Lin and Granger (1993). As this test only examines the in-sample contribution of an additional hidden unit, a second procedure is proposed: Cross-validation, an approach similar to stopped training, considers the out-of-sample contribution of the additional hidden unit.

3.4.1.1 The Test of Lee, White and Granger

The test of Lee, White and Granger (1993), related to the test of Teräsvirta, Lin and Granger (1993), can also be used as a test on additional hidden nonlinearity. Equation (3.1.9) includes here an AR-NN $G(\Theta, X_{t-1})$

$$x_t = G(\Theta, X_{t-1}) + u_t, \quad (3.4.1)$$

with

$$u_t = (F(X_{t-1}) - G(\Theta, X_{t-1})) + \varepsilon_t. \quad (3.4.2)$$

If the first term in equation (3.4.2) is zero, the estimated AR-NN $G(\Theta, X_{t-1})$ does explain the process completely and there is no additional hidden nonlinearity. To test this like in equation (3.1.12) an additional hidden neuron is added to the AR-NN equation:

$$x_t = G(\Theta, X_{t-1}) + \Psi(\gamma_{0a} + \Gamma_a^\top X_{t-1})\beta_a + \varepsilon_t \quad (3.4.3)$$

The index a indicates the additional hidden neuron. The further procedure is now the same as in the nonlinearity test. The artificial linear regression of equation (3.1.17) becomes

$$u_t = \Phi_1 \nabla G(\Theta, X_{t-1}) + \Phi_2 (\Psi(\gamma_{0a} + \Gamma_a^\top X_{t-1})) + u_t^*. \quad (3.4.4)$$

The second term of this equation is approximated by a Taylor polynomial, such that equation (3.4.4) becomes for the *tanh* activation function (like equation (3.1.20)):

$$u_t = \Phi_1 \nabla G(\Theta, X_{t-1}) - \frac{1}{3} \sum_{j_1=1}^n \sum_{j_2=j_1}^n \sum_{j_3=j_2}^n \phi_{2j_1 j_2 j_3} x_{j_1} x_{j_2} x_{j_3} + u_t^* \quad (3.4.5)$$

Thus the null hypothesis can be written as

$$H_0 : \phi_{2_{j_1, j_2, j_3}} = 0 \quad \forall j_1, j_2, j_3 \quad (3.4.6)$$

with alternative

$$H_1 : \phi_{2_{j_1, j_2, j_3}} \neq 0 \quad \forall j_1, j_2, j_3. \quad (3.4.7)$$

The calculation of the test statistics is the same as in subsection 3.1.2.1 (equations (3.1.18) and (3.1.19)). The only difference is the second degree of freedom for the F test statistic, which is here $(T - r)$.

3.4.1.2 Cross Validation

The formal LM test mentioned in the previous subsection only checks if one additional hidden neuron can improve the in-sample performance of a model. But in the empirical part of this dissertation (especially tables 5.10 to 5.13) the main focus is on the out-of-sample performance of AR-NN. It does not always behave in a parallel way to the in-sample performance. Therefore an intuitive method which considers the out-of-sample behavior of the models can be used as an alternative to the formal bottom-up parameter tests (see for example Inoue and Kilian (2006) p.273 for this idea).

The procedure is similar to the one shown in section 3.3.7. The data set is split into a training subset (at which for example stopped training may be applied) and a so called test subset (see Haykin (2009) p.201). Cross validation is executed in three steps:

- In the first step several models with increasing h - starting with $h=0$ - are adjusted to the training subset.
- In the second step the values of the test subset are predicted out of the estimated models. Those values are compared to the original values of the test subset (for example by calculating the RMSE between estimated and original values).
- The model with the lowest test subset RMSE is the optimal model. If the test subset RMSE does not differ significantly for some models, the model with the lowest h (and thus the smallest number of parameters) should be chosen.

3.4.2 Top-Down Parameter Tests

The principle of top-down parameter testing is well known in statistics. The aim of such procedures is to check in an already estimated model if all or certain parameters are significant. Hence the notation top-down describes to start with the estimated model and check if not a smaller model (according to the Occam's razor principle) is better. In this section we propose two types of parameter tests for AR-NN. Those procedures can also be applied at other forms of nonlinear functions (as Taylor-polynomials for example). The first type is an IC. The Neural Network IC (NIC) is a generalization of the AIC and has therefore to be minimized like the AIC. We use this method in accordance to the authors of the original paper (Murata, Yoshizawa and Amari (1994)) to detect the optimal number of hidden neurons. Using the AIC itself would lead to misspecifications. Murata, Yoshizawa and Amari (1994) p.876 cite a Japanese study which shows that problems occur if certain models with different numbers of hidden neurons are compared. Alternatively the NIC can be used to detect the number of input neurons or lags, see Anders (1997) p.77. Then the NIC would belong to section 5.4.1. Another possible alternative would be to vary lags and hidden neurons simultaneously to detect an optimal combination of lags and hidden neurons.

The second type of methods discussed in this section for testing the significance of parameters is one of the classical parameter hypothesis testing procedures in statistical models: The Wald test. The additional feature of this test is the possibility to examine each single γ (as well as α)-weight for significance. Anders (1997) pp.64-73 also mentions the other classical parameter tests, the likelihood ratio (LR) test and the LM test, in context with neural networks, but there are some arguments which speak against them:

The LR test examines two separately estimated models against each other (a restricted and a unrestricted model). This makes only sense if the parameters are estimated by the same algorithm. If a numerical algorithm in combination with stopped training is used, the two models may react differently on the algorithm. Thus here we would compare two differently estimated models, which does not make any sense for the LR test.

Using the LM test as a top-down parameter test means splitting the AR-NN into two parts (the one contains the hidden neurons which should be examined, the other con-

tains the rest) to execute the artificial linear regression with both parts. In particular if the number of hidden units is large, splitting off one or more of them off will lead into a huge programming effort, which is not justified as the Wald test or the NIC examine the same hypothesis and are simpler to implement.

3.4.2.1 Consistency

A prerequisite for parameter tests is the existence of a distribution of the estimated weights, which can be determined at least asymptotical. This requires that the estimators are consistent, which means that they converge to their true value if $T \rightarrow \infty$ (see Anders (1997) p.57). Once consistency of the estimators has been shown it is possible to prove with the help of the central limit theorem the multivariate Gaussian distribution of a standardization of $\hat{\Theta}$ for a sufficient large T (see Widmann (2000) p.53). This distribution is necessary for parameter tests.

A prerequisite of consistency for parameter estimators is that the "true" value of the parameter vector is identified, which means that there exists a unique minimum (not necessarily the global minimum) of the performance function. Particularly for neural networks it might be possible that two different network structures lead to the same minimum. Hence it is absolutely necessary to prespecify the number of hidden neurons h and lags n appropriately, such that no unnecessary parameters are involved (see Widmann (2000) p.54). The lag selection methods as well as the selection of h such that the relation T/r is about 10% might be helpful to avoid overparametrization. Another problem is the order of the hidden units. For any estimated AR-NN with more than one hidden unit the same performance result can be achieved if the order of the hidden units is exchanged. Therefore always two different parameter vectors exist for each minimum. In addition some restrictions or conditions are necessary to guarantee the uniqueness of an optimal parameter vector. The results of White and Domowitz (1984) on the consistency of AR-NN parameters can be recapitulated in the following according to time series specific version of Trapletti, Leisch and Hornik (2000) pp.2431-2434. At first some conditions have to be defined and explained:

Condition 3.1 (Trapletti, Leisch and Hornik (2000) p.2432 assumption 1):

Each hidden neuron contributes nontrivial to the process such that

$\Psi(\gamma_{0j} + \Gamma_j^\top X_{t-1}) \neq 0$ and $\beta_j \neq 0$. All inputs of the hidden neurons are not sign equivalent such that $|\gamma_{0j} + \Gamma_j^\top X_{t-1}| \neq |\gamma_{0k} + \Gamma_k^\top X_{t-1}| \forall j, k = 1, \dots, h$.

This condition should ensure that it is not possible to archive the same equation with fewer hidden units, i.e. that no unnecessary hidden neurons are included. The second part of this condition in particular excludes the twofold existence (with different presign) of any hidden neuron.

Condition 3.2 (Trapletti, Leisch and Hornik (2000) p.2433 assumption 2):

The data generating process is a stationary AR-NN with continuous activation function and the weight space $\Theta \subset \mathbb{R}^r$.

This condition restricts the weight space Θ such that all roots of the characteristic polynomial of the linear part are outside the unit circle (see theorem 2.2). The other weights are restricted to be real numbers.

Condition 3.3 (Trapletti, Leisch and Hornik (2000) p.2434 assumption 3):

$\beta_j > 0 \forall j = 1, \dots, h$ and $\beta_j < \beta_{j+1} \forall j = 1, \dots, h - 1$.

This condition avoids the problem of changing the order of the hidden units as we have discussed before. The restriction of all β_j on positive numbers rules out the possibility of reaching the same value by changing the γ - and β -weight signs. Conditions 1 to 3 restrict the set of possible solutions such that a unique minimum exists for certain activation functions. Those activation functions have to archive the following condition:

Condition 3.4 (Trapletti, Leisch and Hornik (2000) p.2434 assumption 4):

All functions $\Psi(\gamma_{0k} + \Gamma_j^\top X_{t-1})$ and the constant 1 are linear independent $\forall j = 1, \dots, h$ and any $X_{t-1} \subset \mathbb{R}^n$.

The *tanh* activation function fulfills in general this condition.⁸ Theorem 3.1 in the

⁸But if the input is large and always positive/negative, then it would in fact become a constant on the upper bound of its range of values. This is not a realistic case because already in theorem 2.1 it was stated that the weight space should contain a neighbourhood of the origin.

following shows how a NLS-estimator $\hat{\Theta}$ is consistent:

Theorem 3.1 (Trapletti, Leisch and Hornik (2000) p.2434 theorem 5):

If conditions 1-4 are achieved and $E|\varepsilon_t^2|^k < \infty$ for any $k > 1$, the NLS estimator $\hat{\Theta}$ converges almost sure (a.s.) to the true parameter vector Θ , $\hat{\Theta} \xrightarrow{a.s.} \Theta$.

PROOF: For the proof see Trapletti, Leisch and Hornik (2000) p.2439.

Two additional conditions are necessary to calculate the asymptotic normal standardized distribution of $\hat{\Theta}$:

Condition 3.5 (Trapletti, Leisch and Hornik (2000) p.2434 assumption 5):

The original weight vector Θ is in Θ and $\Psi(\cdot)$ is continuously differentiable of order 2.

Condition 3.6 (Trapletti, Leisch and Hornik (2000) p.2434 assumption 6):

All $\Psi'(\gamma_{0j} + \Gamma_j^\top X_{t-1})$, $X_{t-1} \Psi'(\gamma_{0j} + \Gamma_j^\top X_{t-1})$ and the constant 1 are linear independent $\forall j = 1, \dots, h$ and any $X_{t-1} \subset \mathbb{R}^n$.

The *tanh* activation function satisfies condition 5 and 6. Condition 1 and 6 together ensure the regularity of the information matrix at Θ . Theorem 3.2 shows the standardized distribution of $\hat{\Theta}$:

Theorem 3.2 (Trapletti, Leisch and Hornik (2000) p.2434 theorem 6):

If all conditions are achieved, $E|\varepsilon_t^3|^k < \infty$ for any $k > 1$ and $T \rightarrow \infty$, then

$$\frac{1}{\sigma} \sqrt{\nabla^2 Q(\hat{\Theta})} \cdot (\hat{\Theta} - \Theta) \xrightarrow{d} N(0; \mathbf{I}_r), \quad (3.4.8)$$

whereas \xrightarrow{d} denotes convergence in distribution.

PROOF: For the proof see Trapletti, Leisch and Hornik (2000) p.2439

\mathbf{I}_r denotes the r dimensional identity matrix. This result is the basis for the hypothesis

tests in the following section. An alternative representation of equation (3.4.8) using the $(r \times r)$ dimensional covariance matrix C of the parameters, is

$$\sqrt{T}(\hat{\Theta} - \Theta) \xrightarrow{d} N(0; C). \quad (3.4.9)$$

An estimator \hat{C} for C can be received by

$$\hat{C} = \Lambda^{-1} \Upsilon \Lambda^{-1}, \quad (3.4.10)$$

using the crossproduct of the gradient vectors multiplied by T^{-1}

$$\Upsilon = \frac{1}{T} (\nabla Q(\hat{\Theta}))^\top (\nabla Q(\hat{\Theta})) \quad (3.4.11)$$

and the Hessian matrix multiplied by T^{-1}

$$\Lambda = \frac{1}{T} \nabla^2 Q(\hat{\Theta}) \quad (3.4.12)$$

see Murata, Yoshizawa and Amari (1994) p.173 and Onoda (1995) p.278. The relation between the equations (3.4.8) and (3.4.9) is as follows: If the model is appropriate, the crossproduct of the gradient vectors should be equal to the Hessian matrix,

$$\Upsilon = \hat{\sigma}^2 \Lambda. \quad (3.4.13)$$

Consequently equation (3.4.10) is reduced to

$$\hat{C} = \hat{\sigma}^2 \Lambda^{-1}, \quad (3.4.14)$$

see White and Domowitz (1984) p.152. Therefore equation (3.4.8) is the usual transformation of equation (3.4.9) to a standard Gaussian distribution:

$$\begin{aligned} \sqrt{T}(\hat{\Theta} - \Theta) &\xrightarrow{d} N(0; C) \\ C^{-\frac{1}{2}} \sqrt{T}(\hat{\Theta} - \Theta) &\xrightarrow{d} N(0; \mathbf{I}_r) \\ \frac{1}{\sigma} \sqrt{\Lambda T}(\hat{\Theta} - \Theta) &\xrightarrow{d} N(0; \mathbf{I}_r) \end{aligned} \quad (3.4.15)$$

3.4.2.2 The Neural Network Information Criterion

The Neural Network Information Criterion (NIC) is a generalization of the AIC to detect nonsignificant components. It is especially designed for misspecified models, where the usual AIC or BIC can not be used (see Anders (1997) p.77). The AIC in time series analysis is rather used for lag selection than for parameter tests. Hence it is of course possible to use its generalized equivalent, the NIC, also for such aims. Yet this would imply estimating several AR-NN models ex-ante which requires much more effort than to approximate them via the methods of section 3.2 (Taylor polynomials, kernel regression etc.).

First some notations for different AR-NN models have to be introduced. Suggest k neural network functions for the same time series are given, $G_1(\Theta_1, X_{t-1})$, $G_2(\Theta_2, X_{t-1})$, \dots , $G_h(\Theta_h, X_{t-1})$. Let $\Theta_1 \in \mathbb{R}^{r_1}$, $\Theta_2 \in \mathbb{R}^{r_2}$, \dots , $\Theta_h \in \mathbb{R}^{r_h}$ for $r_1 < r_2 < \dots < r_h$. h indicates the originally introduced number of hidden neurons. This means that the number of hidden neurons in the second function is larger than that in the first function and so on. The different models $G_j(\Theta_j, X_{t-1})$, $j = 1, \dots, h$ are not estimated separately. Rather we use the estimated $\hat{\Theta} \approx \Theta$ and delete descending from the largest model $G_h(\Theta_h, X_{t-1})$ the part of the equation as well as the elements of Θ which correspond to the hidden neuron which is not contained in the lower model. Thus we can say that the functions are nested submodels,

$$G_1(\Theta_1, X_{t-1}) \subset G_2(\Theta_2, X_{t-1}) \subset \dots \subset G_h(\Theta_h, X_{t-1}). \quad (3.4.16)$$

Instead of the discrepancy functions proposed in Murata, Yoshizawa and Amari (1994) pp.870-871 we use for each model j the performance function

$$Q(\Theta_j) = \frac{1}{2} \sum_{i=1}^T (x_t - G_j(\Theta_j, X_{t-1}))^2 \quad (3.4.17)$$

known from section 3.3.1. The NIC is defined for model j using Λ_j and Υ_j by

$$NIC_j = \frac{1}{T} \left(\frac{1}{2} \sum_{t=1}^T \varepsilon_{jt}^2 + tr(\Upsilon_j \Lambda_j^{-1}) \right). \quad (3.4.18)$$

The second term is called penalty term because it "punishes" the excessive usage of parameters. Should the model j be faithful, which means that Θ_j is a (optimal) NLS

solution, we know from equation (3.4.13) that $\Upsilon_j = \sigma_j^2 \Lambda_j$. A consequence would be that

$$\text{tr}(\Upsilon_j \Lambda_j^{-1}) = \sigma^2 \cdot \text{tr}(\mathbf{I}) = \sigma^2 r \quad (3.4.19)$$

Thus in this case the *NIC* is equal to the *AIC*₂ (equation (3.2.2)). Like any other IC the *NIC* has to be minimized, thus we choose the submodel with the lowest *NIC*.

The *NIC*, like any other IC, is not consistent. If IC in general are used as parameter tests, it has to be ensured that no irrelevant hidden neurons are included in the models (see Anders (1997) p.78). If the models are overparametrized, Υ_j and Λ_j are degenerated and may diverge heavily. The consequence is an unusual large *NIC* value. Hence large *NIC*'s can be interpreted as a sign for overparametrization (see Anders (1997) p.79).

3.4.2.3 The Wald Test

Before the Wald test (rooted in Wald (1943)) can be executed, it is necessary to estimate the covariance matrix *C* using equation (3.4.10). In addition no irrelevant hidden neurons should be included in the models. For the following see Widmann (2000) pp.100-102 and Anders (1997) pp. 72-73. The simplest application of the Wald test is to consider a null hypothesis of non-significance for each single weight,

$$H_0 : \theta_i = 0 \quad \forall i = 1, \dots, r. \quad (3.4.20)$$

In this case the test statistic is

$$T_{WALD_1} = \frac{\theta_i^2}{\sigma_{\theta_i}^2}. \quad (3.4.21)$$

The denominator $\sigma_{\theta_i}^2$ is the variance of θ_i . This equals the *i*th element on the principal diagonal of *C*. The test statistic is χ^2 distributed with one degree of freedom (see Anders (1997) p.73). Davidson and MacKinnon (1993) p.89 call it "pseudo- t-statistic".

4 Multivariate models

4.1 Multivariate AR-NN

4.1.1 Vector Autoregressive Neural Network Equations

Our knowledge of neural network modelling of autoregressive processes now is easily extendable to nonlinear vector autoregressive models. In general multivariate modelling means to add an additional dimension - the number of variables m - to a univariate model. The idea of multivariate AR-NN is based on Raman and Sunlikumar (1995) and Chakraborty et al. (1992), although vector representation as we will use it in this section is not treated by those authors. A vector autoregressive process is introduced formally (equivalent to definition 2.1) by

Definition 4.1 (Vector autoregressive process (VAR)):

A process is called vector autoregressive process of order n - short VAR(n) - if it is described by the functional relation

$$Y_t = F(\mathbf{Y}_{t-1}) + E_t, \quad (4.1.1)$$

whereas $Y_t = (y_{1t}, y_{2t}, \dots, y_{mt})$ is a vector of m variables and \mathbf{Y}_{t-1} is a $(m \cdot n \times 1)$ vector of the lagged variables,

$$\mathbf{Y}_{t-1} = (Y_{t-1}, Y_{t-2}, \dots, Y_{t-n})^\top \quad (4.1.2)$$

with $F : \mathbb{R}^{mn} \rightarrow \mathbb{R}^m$ and $E_t = (\varepsilon_{1t}, \varepsilon_{2t}, \dots, \varepsilon_{mt})^\top$ is a m -dimensional Gaussian distributed WN vector.

Remark 4.1.1:

If $F(Y_{t-1})$ is a linear function, the process is a linear VAR. If $F(Y_{t-1})$ is nonlinear it is a nonlinear VAR.

A linear VAR is represented by

$$Y_t = A_0 + A_1 Y_{t-1} + A_2 Y_{t-2} + \dots + A_n Y_{t-n} + E_t \quad (4.1.3)$$

whereas the constant matrix A_0 has dimension $(m \times 1)$ and the parameter matrices A_i $\forall i = 1, \dots, n$ have dimension $(m \times m)$. An alternative representation of equation (4.1.3) is

$$Y_t = A_0 + \underbrace{\mathbf{A}}_{(m \times m \cdot n)} \underbrace{\mathbf{Y}_{t-1}}_{(m \cdot n \times 1)} + E_t, \quad (4.1.4)$$

whereas $\mathbf{A} = (A_1, A_2, \dots, A_n)$. In full matrix representation equation (4.1.4) is written as follows:

$$\begin{pmatrix} y_{1,t} \\ y_{2,t} \\ \vdots \\ y_{m,t} \end{pmatrix} = \begin{pmatrix} \alpha_{01} \\ \alpha_{02} \\ \vdots \\ \alpha_{0m} \end{pmatrix} + \begin{pmatrix} \alpha_{111} & \dots & \alpha_{11m} & \dots & \alpha_{n1m} \\ \alpha_{121} & \dots & \alpha_{12m} & \dots & \alpha_{n2m} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \alpha_{1m1} & \dots & \alpha_{1mm} & \dots & \alpha_{nmm} \end{pmatrix} \begin{pmatrix} y_{1,t-1} \\ \vdots \\ y_{1,t-n} \\ \vdots \\ y_{2,t-1} \\ \vdots \\ y_{m,t-n} \end{pmatrix} + \begin{pmatrix} \varepsilon_{1t} \\ \vdots \\ \varepsilon_{mt} \end{pmatrix}$$

By the independence assumptions of the residuals one can divide (4.1.4) into m independent linear equations

$$y_{j,t} = \alpha_{0j} + \alpha_{1j1} y_{1,t-1} + \dots + \alpha_{1jm} y_{m,t-1} + \alpha_{2j1} y_{1,t-2} + \dots + \alpha_{njm} y_{m,t-n} + \varepsilon_{jt} \quad (4.1.5)$$

for each variable $j = 1, \dots, m$. The separate models for each output neuron have the same dimensional properties like univariate models. The main difference is, that there are some additional variables which might be Granger-causal to the output variable. Informally in times series context a variable y_2 is said to Granger-cause any other variable y_1 , if its lagged past values significantly contribute to explain the present state of y_1

(see for example Granger (1988) p.203). The neural network modelling procedure discussed here is equivalent to a linear VAR like equation (4.1.3). In particular the property to split the multivariate model into several independent univariate models (like in equation (4.1.5)) is essential. Let us introduce the following additional matrix terms for formulation of the Vector AR-NN (VAR-NN):

$$\Gamma_{0,j} = \begin{pmatrix} \gamma_{0j1} \\ \gamma_{0j2} \\ \vdots \\ \gamma_{0jm} \end{pmatrix}$$

$$\Gamma_{i,j} = \begin{pmatrix} \gamma_{ij11} & \gamma_{ij12} & \cdots & \gamma_{ij1m} \\ \gamma_{ij21} & \gamma_{ij22} & \cdots & \gamma_{ij2m} \\ \vdots & \vdots & \vdots & \vdots \\ \gamma_{ijm1} & \gamma_{ijm2} & \cdots & \gamma_{ijmm} \end{pmatrix}$$

$$\mathbf{\Gamma}_j = (\Gamma_{1j}, \Gamma_{2j}, \dots, \Gamma_{nj})$$
(4.1.6)

A VAR-NN is thus defined by the multivariate version of equation (2.2.7):

$$\underbrace{Y_t}_{(m \times 1)} = \underbrace{A_0}_{(m \times 1)} + \underbrace{\mathbf{A}}_{(m \times mn)} \underbrace{Y_{t-1}}_{(mn \times 1)} + \sum_{j=1}^h \Psi \left(\underbrace{\Gamma_{0j}}_{(m \times 1)} + \underbrace{\mathbf{\Gamma}_j}_{(m \times mn)} \underbrace{Y_{t-1}}_{(mn \times 1)} \right) \underbrace{\beta_j}_{1 \times 1} + \underbrace{E_t}_{m \times 1} \quad (4.1.7)$$

In this case $\Psi : \mathbb{R}^m \rightarrow \mathbb{R}^m$ whereas each element is transformed individually by the activation function (the tanh for example). Like in the linear case (equation (4.1.5)), m independent models can be splitted off. Note that the parameter β_j is scalar (1×1) to keep the structure of the NN-VAR straightforward. In addition β_j has to be prespecified, because it can not be estimated simultaneously with the other parameters.

Stopped training becomes more complicated in the multivariate models, as for estimation the equations have to be separated, but for forecasting they have to be added together. The problem is, that for forecasting one variable, one also needs the estimated

models for the other variables. One may proceed as follows: Firstly the parameter vectors are estimated for each variable and saved for each iteration. Then the multivariate model is used to evaluate all combinations of the parameters in each iteration. The optimal model minimizes the VS-RSS for each variable. Such a procedure requires much effort, especially for implementing all the combinations of parameter vectors. Hence we recommend to abstain from stopped training in the multivariate models.

One has to overcome a difficulty if the lag selection methods from section 3.2 are applied at VAR-NN. The lag selection methods can be applied at the submodels of the VAR-NN without problems but the result are certainly different lag structures. Merging the submodels to a VAR-NN is then impossible. Thus a procedure has to be implemented which detects the optimal lag for the whole AR-NN. The following steps might provide a solution:

- Calculate the lag selection criterion for several lags for the submodels (especially for the methods in 3.2.3 and 3.2.4 the submodels can be approximated by Taylor expansion or local linear estimation as well as the univariate models).
- Sum the lag selection criteria of all submodels for each lag.
- Choose that lag where the sum of lag selection criteria is minimal.

The last step results of the fact that the lag selection criteria in general have to be minimized. The parameter estimation methods may in a similar way also be used simultaneously for the submodels. But they also allow for partly weakly exogeneity of some variables if parts of the multivariate hidden neurons are not significant. Alternatively the lag order can be detected by the average lag of the univariate time series optimal lag orders.

4.1.2 Vector Autoregressive Neural Network Graphs

To start with graph representation of a VAR-NN, we first consider the graph of a linear VAR(2) with two variables using the symbols from section 2.2.1 with white circles for the first variable and yellow circles for the second variable (see figure 4.1). The output neurons in this graph can be considered separately (see figure 4.2 for the separated first variable). In the following we draw the graph of a VAR-NN(2) with two variables. Before we proceed with the multivariate AR-NN we have to introduce some additional

symbols, see table 4.1. Note that here the β -weights connect vectors, but they apply to each single variable in the vectors. This is important if the VAR-NN are separated for estimation. Now the hidden layer is inserted into the linear AR(2) model of figure 4.1. Figure 4.3 shows the "black box" representation of a VAR-NN(2), in figure 4.6 the "black box" is unveiled. Alternatively the variable can be hidden to simplify the graph to a vector representation (see figure 4.7).




| Symbol | Formal |
|---|-----------|
|  | y_1 |
|  | y_2 |
|  | β_i |

Table 4.1: Additional symbols for a 2 variable VAR-NN

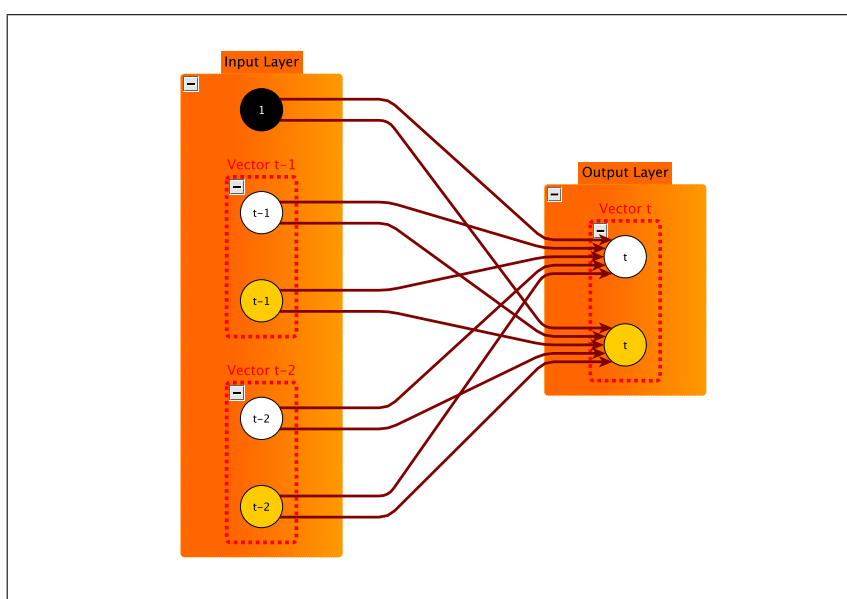


Figure 4.1: VAR(2) graph with 2 variables

Source: Authors' design

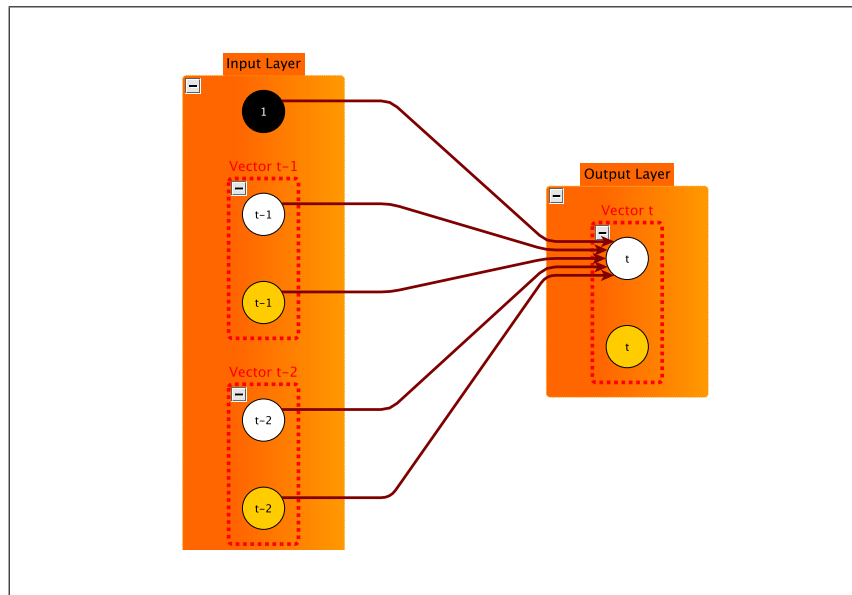


Figure 4.2: Separated model of the first variable

Source: Authors' design

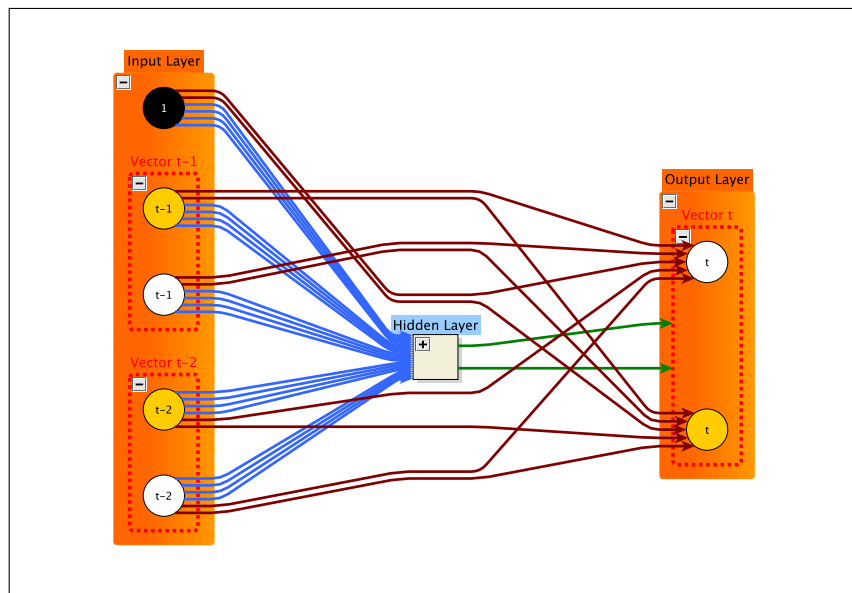


Figure 4.3: VAR-NN(2) - "black box" representation

Source: Authors' design

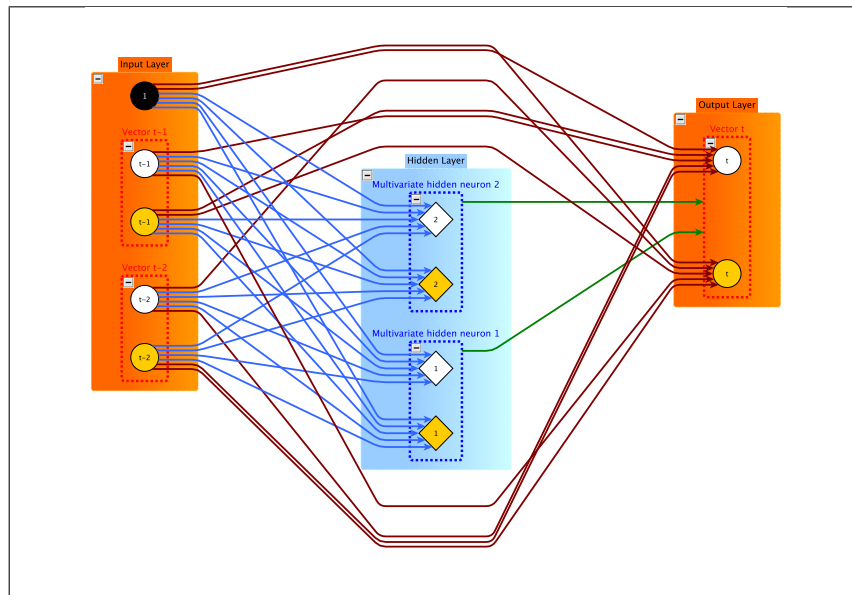


Figure 4.4: VAR-NN(2) graph

Source: Authors' design

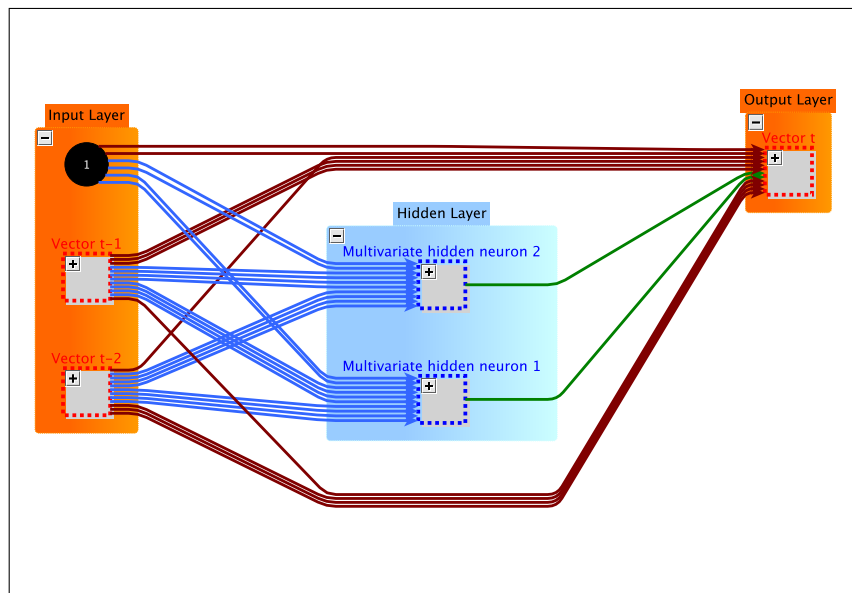


Figure 4.5: VAR-NN(2) - vector representation

Source: Authors' design

4.2 Neural Networks and Cointegration

The concept of cointegration has become essential in time series analysis since Engle and Granger (1987). In general the term cointegration is defined by (see for example Johansen (1995) p.35):

Definition 4.2 (Cointegration):

If for a vector Y_t of m series of the same integration order ≥ 1 a stationary linear combination $B^\top Y_t$, exists, the series are called cointegrated with cointegration matrix B of dimension $(c \times m)$ with $c < m$. If $c=1$, B is called cointegration vector.

In this section we will show how cointegration can be related to our AR-NN models. Note that this section is not about nonlinear cointegration but rather about nonlinear adjustment in vector error-correction models (VEC), see therefore Dufrenot and Mignon (2002) p.224.

4.2.1 Nonlinear Adjustment in Error Correction Models

First of all we consider a linear VEC in reduced rank representation as for example in Johansen (1995) p.45:

$$\Delta Y_t = KB^\top Y_{t-1} + A_0 + A_1 \Delta Y_{t-1} + \dots + \Delta Y_{t-n} + E_t \quad (4.2.1)$$

B is the $(m \times c)$ cointegration matrix with $0 \neq c < m$ and K is the $(m \times c)$ loading matrix. c is the number of cointegration relationships. For details see Johansen (1995) pp.45-69. The cointegration matrix influences the variables via K through a linear relationship. If there is nonlinearity in the data, nonlinear instead of linear adjustment might improve the model (4.2.1). Using the nonlinear function $F(\cdot)$ equation (4.2.1) can be rewritten as

$$\Delta Y_t = F(B^\top Y_{t-1}) + A_0 + A_1 \Delta Y_{t-1} + \dots + \Delta Y_{t-n} + E_t. \quad (4.2.2)$$

Note that here $F : \mathbb{R}^c \rightarrow \mathbb{R}^m$ is the nonlinear equivalent to K in equation (4.2.1). This VEC can be illustrated by a simple bivariate example from Escribano and Mira (2002) p.514:

$$\Delta x_t = \alpha_{11} \Delta x_{t-1} + f_1(z_{t-1}) + v_{1t} \quad (4.2.3)$$

$$\Delta y_t = \alpha_{21} \Delta y_{t-1} + f_2(z_{t-1}) + v_{2t} \quad (4.2.4)$$

$$z_t = x_t - by_t \quad (4.2.5)$$

z_t is the cointegration relationship and the cointegration vector here is $B = (1, b)^\top$. $f_1(\cdot)$ and $f_2(\cdot)$ are nonlinear functions mapping $F_1, F_2 : \mathbb{R} \rightarrow \mathbb{R}$, $Y_t = (x_t, y_t)^\top$,

$$F(B^\top Y_{t-1}) = \begin{pmatrix} f_1(z_{t-1}) \\ f_2(z_{t-1}) \end{pmatrix} \quad (4.2.6)$$

and

$$A = \begin{pmatrix} \alpha_{11} & 0 \\ \alpha_{21} & 0 \end{pmatrix} \quad (4.2.7)$$

4.2.1.1 Theoretical Prerequisites

Now should be shown how such a nonlinear VEC is theoretically justified. We will therefore use theorem 3.7 in Escribano and Mira (2002) p.517. For the formulation two new expressions have to be introduced, which both are closely related to stationarity: α -mixing and near epoch dependency (NED). With those constructs a functional central limit theorem holds for the nonlinear error correction theorem. Such a functional central limit theorem is the basis for estimation and inference (see Escribano and Mira (2002) p.511-512). For this section see Escribano and Mira (2002) pp.512-515.

Beforehand some mathematical notations have to be explained: Let Ω be the set of all possible realizations of a process x_t . The σ -algebra \mathcal{F} of Ω is then defined as a system of subsets which

- contains the null set
- contains for each subset $\mathcal{A} \in \mathcal{F}$ also the complement $\bar{\mathcal{A}}$ of \mathcal{A}
- contains for a (possibly) infinite ($i \rightarrow \infty$) sequence of sets \mathcal{A}_i also their union $\bigcup_i \mathcal{A}_i$

see Hassler (2007) p.14. The σ -algebra $\mathcal{F}_s^t = \sigma(x_s, \dots, x_t)$ generated by a process x_t is defined as the smallest σ -algebra for which x_t is measurable. Therewith the mixing coefficients α are defined as

$$\alpha^k = \sup_t \sup_{\{F_1 \in \mathcal{F}_{-\infty}^t, F_2 \in \mathcal{F}_{t+k}^\infty\}} |Prob(F_1 F_2) - Prob(F_1) Prob(F_2)| \quad (4.2.8)$$

sup denotes the supremum (upper bound) of a set. We call the process α - or strong-mixing if $\alpha^k \rightarrow 0$ as $k \rightarrow \infty$. A strong-mixing process can be interpreted as a process, where the dependence between two realizations of the process x_t , which are separated by k steps, decreases as k increases. The mixing coefficient α^k measures this dependence. A stationary process is strongly mixing (but not vice versa).

The second concept necessary for the nonlinear error correction theorem is the NED. Let y_t be a process with a finite sum of squares ($E(y_t^2) < \infty$). y_t is NED of size $-a$ on the process x_t if

$$\phi(n) = \sup_t \|y_t - E(y_t | x_{t-n}, \dots, x_{t+n})\|_2 \quad (4.2.9)$$

is of size $-a$. $\|\cdot\|_2$ is the L_2 -norm $E^{\frac{1}{2}}|\cdot|^2$. To keep this concept in accordance with AR theory, we assume that the forward values of $x_t, x_{t+i} \forall i = 1, \dots, n$ do not improve the conditional expectation and are therefore useless. If $\phi(n)$ goes to zero as n increases, it can be said that y_t essentially depends on the recent epoch of x_t . y_t is NED of any size if it depends on a finite number of lags of x_t .

The NED property is important to characterize an $I(0)$ -process via a functional central limit theorem for NED variables. The functional central limit theorem for NED processes explains how a standardization of an $I(0)$ NED process converges to a standard Brownian motion and justifies thus the distribution of such a process.

Theorem 4.1 (Escribano and Mira (2002) p.513 theorem 2.3):

Let x_t be a process with zero mean, uniformly L_s -bounded and NED of size $-\frac{1}{2}$ on an α -mixing process of size $-s/(s-2)$ and $T^{-1}E(\sum_{t=1}^T x_t)^2 \rightarrow \sigma^2$, $\sigma^2 \in]0, \infty[$. Then $T^{-\frac{1}{2}} \sum_{t=1}^{\lfloor Ts \rfloor} x_t$ converges to a standard Brownian motion $B(s)$.

A process y_t is $I(0)$ if it is NED on a mixing process x_t , but the process $w_t = \sum_{s=1}^t y_s$ is not NED on x_t . If $w_t = \sum_{s=1}^t y_s$ is NED on x_t , the process is $I(1)$. Similarly we can express linear bivariate cointegration in terms of NED: Two $I(1)$ processes x_t and y_t are linearly cointegrated with cointegration vector $B = (1, -b)^\top$ if $x_t - by_t$ is NED on a strong-mixing process, but $x_t - \delta y_t$ is not NED on a strong mixing process for any $\delta \neq b$.

In the following we will use the nonlinear VAR

$$Z_t = H(Z_{t-1}) + U_t \quad (4.2.10)$$

with dimensions of Z_t and U_t ($c \times 1$). $H(\cdot) : \mathbb{R}^c \rightarrow \mathbb{R}^c$ is a differentiable function of a variable Z on an open set of \mathbb{R}^m . Under the following assumptions and conditions is Z_t NED on the α -mixing sequence U_t :

- U_t is α -mixing of size $-s/(s-2)$ for $s > 2$.
- $SR(J_Z(H(Z))) \leq 1 - \delta$, whereas $SR(\cdot)$ denotes the spectral radius of a matrix, which is its largest eigenvalue. $J_Z(H(Z))$ denotes the Jacobi matrix of $H(Z)$ with respect to Z . Its dimension is $(c \times c)$. This condition is also called boundedness condition, because it limits the largest eigenvalue of the Jacobian matrix away from 1. The boundedness condition is a sufficient condition for $H(Z_{t-1})$ to be NED (see Escribano and Mira (2002) p.516).
- Let Δ_u be a finite constant and $E\|U_t\|_S^2 = \Delta_u$. $\|\cdot\|_S$ denotes a subordinate matrix norm.

4.2.1.2 The Nonlinear Error Correction Model and Neural Networks

Using the notations from the section above, we can formulate the VEC-theorem of Escribano and Mira (2002) p.517. Note that the theorem is formulated for only one lag ($n = 1$) in the VAR-part:

$$\Delta Y_t = A_1 \Delta Y_{t-1} + F(B^\top Y_{t-1}) + E_t \quad (4.2.11)$$

Theorem 4.2 (Escribano and Mira (2002) p.517 theorem 3.7):

We consider the VEC in equation (4.2.11). Assume that

- E_t is α -mixing of size $-s/(s-2)$ for $s > 2$
- $\sum_{t=1}^T E_t$ is not mixing on a NED sequence
- $E \|E_t\|_S^2 \leq \Delta_u$
- $F(B^\top Y_{t-1}) = F(Z_{t-1})$ with $Z_t = B^\top Y_t$ of dimension $(c \times 1)$ and $F(\cdot)$ is a continuously differentiable function fulfilling the general Lipschitz conditions (see Escribano and Mira (2002) p.7 for details)
- $SR(A_1) < 1$
- For some $\delta \in [0; 1]$

$$SR = \underbrace{\begin{pmatrix} \underbrace{A_1}_{(m \times m)} & \underbrace{J_Z(F(Z_t))}_{(m \times c)} \\ \underbrace{B^\top A_1}_{c \times m} & \underbrace{I_c + B^\top J_Z(F(Z_t))}_{(c \times c)} \end{pmatrix}}_{(m+c) \times (m+c)} \leq 1 - \delta \quad (4.2.12)$$

Then ΔY_t and Z_t are simultaneously NED on a α -mixing sequence U_t whereas $U_t = B^\top E_t$ and Y_t is $I(1)$.

PROOF: For the proofs we refer to the original literature (see Escribano and Mira (2002) pp.517-518).

The AR-NN function fulfills the general Lipschitz conditions as it consists of one linear and several bounded functions. A VEC-NN with more than $n = 1$ lags can be brought in accordance with the theorem by some rearrangement:

$$\underbrace{\Delta Y_t}_{(m \cdot n \times 1)} = \underbrace{A_1}_{(m \cdot n \times m \cdot n)} \cdot \underbrace{\Delta Y_{t-1}}_{(m \cdot n \times 1)} + F(B^\top Y_{t-1}) + E_t \quad (4.2.13)$$

with $\mathbf{Y}_t = (Y_t, \dots, Y_{t-n})^\top$, $\mathbf{Y}_{t-1} = (Y_{t-1}, \dots, Y_{t-n-1})^\top$, $\mathbf{B} = (\underbrace{B, \dots, B}_{n \text{ times}})^\top$, $F: \mathbb{R}^r \rightarrow \mathbb{R}^{m \cdot n}$, $\mathbf{E}_t = (E_t, \dots, E_{t-n})^\top$ and

$$\mathbf{A}_1 = \begin{pmatrix} A_1 & \dots & A_n \\ \vdots & \vdots & \vdots \\ A_1 & \dots & A_n \end{pmatrix}$$

However for estimation of the nonlinear VEC we may use the usual form (equation 4.2.2). We can formulate an example for a bivariate nonlinear VEC with an AR-NN representation of $F(Z_{t-1})$ and one cointegration relationship ($c = 1$). Let $n = 1$ and $h = 1$ (see also Dufrenot and Mignon (2002) pp.229-243 for other applications of the VEC of Escribano and Mira (2002)):

$$z_t = y_t - bx_t \quad (4.2.14)$$

$$\begin{aligned} \Delta x_t &= \alpha_{11}^* \Delta x_{t-1} + \alpha_{12}^* \Delta y_{t-1} + \\ &\alpha_{01} + \alpha_{11} z_{t-1} + \Psi(\gamma_{01} + \gamma_{11} z_{t-1}) \beta + \varepsilon_{1t} \end{aligned} \quad (4.2.15)$$

$$\begin{aligned} \Delta y_t &= \alpha_{12}^* \Delta y_{t-1} + \alpha_{22}^* \Delta x_{t-1} + \\ &\alpha_{02} + \alpha_{12} z_{t-1} + \Psi(\gamma_{02} + \gamma_{12} z_{t-1}) \beta + \varepsilon_{2t} \end{aligned} \quad (4.2.16)$$

A vector representation of equations (4.2.14), (4.2.15) and (4.2.16) together is given in the following:

$$\begin{aligned} \begin{pmatrix} \Delta x_t \\ \Delta y_t \end{pmatrix} &= \begin{pmatrix} \alpha_{11}^* & \alpha_{12}^* \\ \alpha_{21}^* & \alpha_{22}^* \end{pmatrix} \begin{pmatrix} \Delta x_{t-1} \\ \Delta y_{t-1} \end{pmatrix} + \\ &\begin{pmatrix} \alpha_{01} \\ \alpha_{02} \end{pmatrix} + \begin{pmatrix} \alpha_{11} \\ \alpha_{21} \end{pmatrix} z_{t-1} + \Psi \left(\begin{pmatrix} \gamma_{01} \\ \gamma_{02} \end{pmatrix} + \begin{pmatrix} \gamma_{11} \\ \gamma_{12} \end{pmatrix} z_{t-1} \right) \beta \\ &+ \begin{pmatrix} \varepsilon_{1t} \\ \varepsilon_{2t} \end{pmatrix} \end{aligned} \quad (4.2.17)$$

4.2.2 NN-VEC graphs

The NN-VEC can also be depicted as a graph. In the following we assume a NN-VEC with 3 variables, 2 lags and 2 hidden neurons in the nonlinear adjustment of the cointegration relationship. Such a VEC is given by equation

$$\widehat{\Delta Y}_t = A_1^* \Delta Y_{t-1} + A_2^* \Delta Y_{t-2} + A_0 + A_1 z_{t-1} \sum_{j=1}^2 \Psi(\Gamma_{0j} + \Gamma_{1j} z_{t-1}) \beta_j \quad (4.2.18)$$

with $y_t = (y_{1t}, y_{2t}, y_{3t})^T$ and $z_{t-1} = (b_1, b_2, b_3)^T Y_{t-1}$. We need some additional symbols, shown in table 4.2. The linear cointegration relationship is shown in figure 4.6 and the nonlinear VEC-NN in figure 4.7.








| Symbol | Formal | Symbol | Formal |
|---|--------|---|--------------|
|  | y_1 |  | Δy_1 |
|  | y_2 |  | Δy_2 |
|  | y_3 |  | Δy_3 |
|  | z | | |

Table 4.2: Additional symbols for a 3 variable NN-VEC

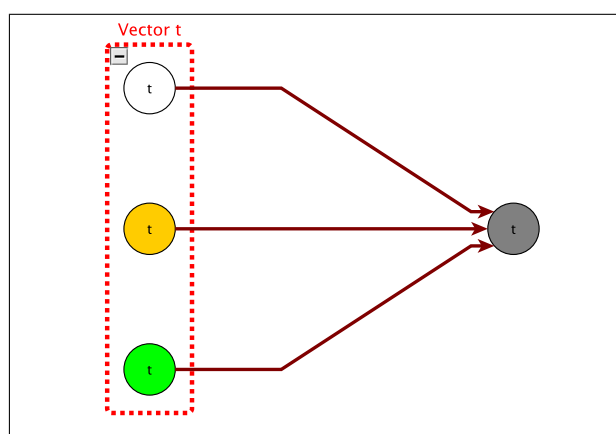


Figure 4.6: Linear cointegration relationship (3 variables)

Source: Authors' design

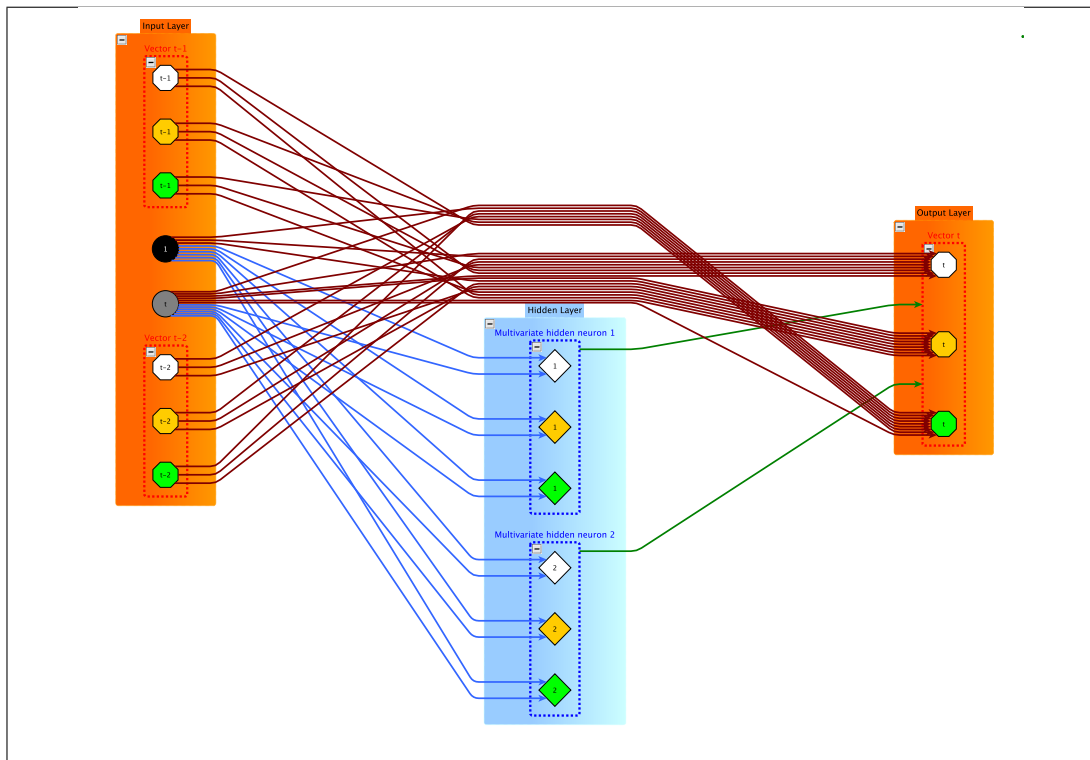


Figure 4.7: NN-VEC with 2 lags, 3 variables and 2 hidden neurons

Source: Authors' design

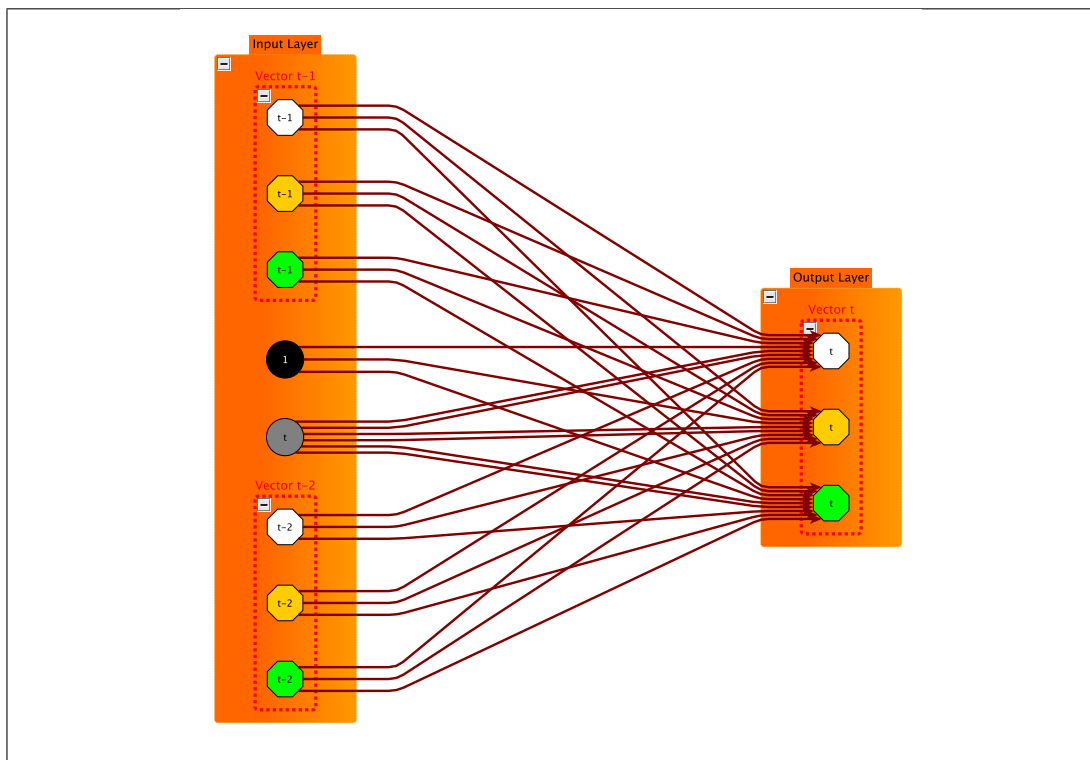


Figure 4.8: Linear VEC with 2 lags, 3 variables

Source: Authors' design

4.2.3 Identifying and Testing the NN-VEC

The nonlinear VEC of Escribano and Mira (2002) in combination with AR-NN from the previous section (NN-VEC) is identified via several steps. Partly we follow Dufrenot and Mignon (2002) p.229 here. The steps are summarized in the following:

- Identify the linear cointegration relationship $B^T Y_{t-1}$. The easiest way is to use a two or more stage least squares procedure (2SLS, 3SLS). Using the 2SLS procedure to detect bivariate cointegration is for example discussed in Al-Ballaa (2005). Therefore a structural equation system has to be constructed using the given variables. One or more variables included in the structural equation system affect the cointegration relationship only indirectly via estimation of the instrumental variable (they are not included in the cointegration relationship). A possible field of application are supply-demand equations with nonlinear variables (supply and demand observed over time are nonlinear). In the VEC-NN example from section 4.2.1.2 the result is an estimator for equation (4.2.14).
- Test the cointegration relationship on the α -mixing property. As there exist no formal tests of α -mixing, tests on stationarity are sufficient.
- Estimate the NN-VEC using the Levenberg-Marquardt algorithm. Therefore the m -dimensional NN-VEC has to be split into m equations which map on the univariate \mathbb{R} space (one equation for each variable). After this step the results in the VEC-NN example from section 4.2.1.2 are the equations (4.2.15) and (4.2.16). They can be brought together to receive the multivariate model (4.2.17).
- Test the nonlinear part of the nonlinear adjustment in the VEC on significance. It might be difficult to compute the Wald statistic or the NIC for large m . A simpler way, which in addition is focused on the out of sample behavior, is cross validation (see section 3.4.1.2). Even a cointegration test can be executed using cross validation: A linear VAR as well as NN-VEC with $h = 0, \dots$ are estimated (repeating step three for various h). If the out-of-sample performance of the VAR is significantly better than any NN-VEC, there is evidence for no cointegration.

Finally some remarks to the test on the mixing property in the cointegration relationships: Tests on stationarity of the residuals are in general sufficient to guarantee the α -mixing property. By definition α -mixing includes stationary processes. Dufrenot and

Mignon (2002) use a response- surface (R/S) test in addition, but this does not contribute any further findings. Therefore we will only consider stationarity in the empirical part.

5 The German Automobile Industry and the US Market

The aim of this study is to predict some variables (macroeconomic, industry specific and financial) connected with the German automobile industry using univariate AR-NNs and a NN-VEC with one cointegration relationship. We use data from a time period including the crisis of 2008/2009, which certainly is responsible for nonlinearities in the data set. Mishkin (1996) p.17 defines a financial crisis as a nonlinear disruption to financial markets.¹ Such a definition restricted to financial markets can easily be transferred to the real economy, in our case the German automobile industry. We observe that data without crisis tend to be linear whereas data including a crisis tend to be nonlinear (tables 5.2 and 5.4). Thus, especially for data including a crisis nonlinear methods are necessary.

Our focus is particularly on the relations between the German automobile industry and the US market. The macroeconomic variable is the exchange rate between US Dollar and Euro (USD/EUR). We choose two industry specific variables: The first, the industrial production of car manufacturers in Germany, can be considered as a supply variable. The second, imports of foreign cars to the USA, serves as a demand variable. As financial variable we use an index of the stock prices of three German car manufacturing companies. Several recent and past news (see for example Moody's (2008)) emphasise the special relationship between the German automobile manufacturers and the US car market including the mentioned variables.

For the analysis of the data we first construct univariate AR-NN models for all variables, after we have shown that all series are nonlinear by the test of Lee, White and Granger (1993) and integrated of order one by the RADF-test. We execute all the variable selection procedures and parameter tests from chapter 3. The Levenberg-Marquardt algorithm is used for parameter estimation. The resulting models are com-

¹Although this paper is about asymmetric information, this statement of the definition affects financial time series as well.

pared to several linear AR and ARMA models. Afterwards a cointegrated model is adjusted, whereas the cointegration relationship is calculated using a structural equation model and 2SLS regression. The cointegration relationship includes the variables industry production of the German automobile industry, sales of foreign cars in the USA and the exchange rate. The stock price index flows indirectly into the cointegration relationship, via estimation of the instrumental variable. We consider this financial variable only as a management incentive, which indirectly exercises some influence on industry production and sales. For calculations, some of the functions are implemented in the statistical programming language R. The code is provided in appendix B. As far as possible, some already existing R-packages as well as the software JMulti are used.

5.1 Economic Motivation

Before we proceed with the statistical analysis of the data, the economic motivation behind the statistical models, especially a cointegrated model, should be explained. Therefore at first the economic meaning of the technical terms in the NN-VEC is considered: In the cointegrated model we distinguish between long-run and short-run effects. The cointegration relationship, often referred to as long-run-equilibrium, represents the long run effects, whereas the VAR part of the NN-VEC describes the short run effects. Long- and short-run effects are combined by the NN-VEC. Intuitively we assume that the univariate AR-NN are only able to describe short-run effects, as they do not include any long-run economic relationship. This assumption is confirmed by our results below.

A simple example should show how the long run relationship between industry production, sales of foreign cars in the USA and the exchange rate are expected to explain some economic developments: A long-run increase of the exchange rate, caused by a weak USD, leads to lower sales of foreign cars in the USA and consequently a lower industry production in the German car manufacturing industry. This statement is only valid for the German premium car manufacturers (the diversified VW corporation is not much affected as its production for the US market is located in the NAFTA area and its cars don't count as imported). In other words we consider the exchange rate exposure of the German premium automobile industry with focus on sales and industry production. Some statistics underline how important the US market for German car manufacturers is (see VDA (2010)): In 2008 74.4% and in 2009 68.8% of the production of German

car manufacturers was exported, the USA was the most important non-EU market (accounting in 2008 for 12.6% and in 2009 for 10.5% of the total German car export). An increasing exchange rate caused by a weakening USD may have the following two implications on the German automobile industry:

- The German car manufacturers increase the price of their products in the USA to keep the profit constant. On the demand side US consumers observe the persistent increase in prices of foreign cars. Consequently they look for alternatives. Hence the industry production of the German car manufacturers decreases, as they lose their share of the important US automobile consumer market (Humboldt Institution on Transatlantic Issues (2005) p.3). Adjustment to the equilibrium is certainly a long-term issue, as cars are not goods of daily consumption and it takes some time until the increase in the purchase price of the cars reaches the consumer.
- Car manufacturers keep the prices constant. Consequently their profit decreases, because of the increasing production costs (for production in the Euro currency area) measured in USD. To increase the profit again, production has to be shifted to the USD currency area. Intuitively, this strategy seems superior to the first one, because it does not imply that the car manufacturers accept the reduction of profit. In fact it was reported by the media September 2009 that BMW plans to invest additional 750 million EUR in its US plant (see Handelsblatt (2009a)) and in December 2009 that Daimler plans to shift production of the Mercedes C-class partly to the USA (see Handelsblatt (2009b)). Even earlier BMW responded on an assumed long-run weak USD by enlarging its US based production (Harbour and Joas (2008) p.67). Hence the cars produced in the USA do not count any more as imported foreign cars, with the consequence that the number of sales of imported foreign cars decreases. Subsequently, the industrial production of the German car manufacturing industry also decreases.

Both chains of reaction lead to the same result: The long-run-equilibrium between the three variables is maintained. A vice versa effect appears if the USD is strong as in the years before 2003 (Mohatarem (2003)). Relationships between the variables are shown in figure 5.1: The relation between EXC and IND and SAL respectively is negative. In the NN-VEC the additional nonlinear term involving the cointegration relationship accounts for the fact, that this linear equilibrium affects nonlinear time series and has

to be processed nonlinear.

The stock index as a financial variable is involved in the cointegration relationship indirectly via the calculation of the instrumental variable. It is used as a fourth variable, because the stock market price can serve as an incentive for management actions. For example, only changes in the share prices caused by decreasing sales, forces the management to intervene.

For prediction of the individual time series not only the long-run-equilibrium is necessary. In addition, the model has to take into consideration the short run deviations from the equilibrium. They occur, if the series involved in the model are affected by a short-run change, which disappears over time and has no relation to the long-run-equilibrium. The linear VAR as a part of the NN-VEC (first term on the left hand of the NN-VEC equation) tries to explain such short-run-effects.

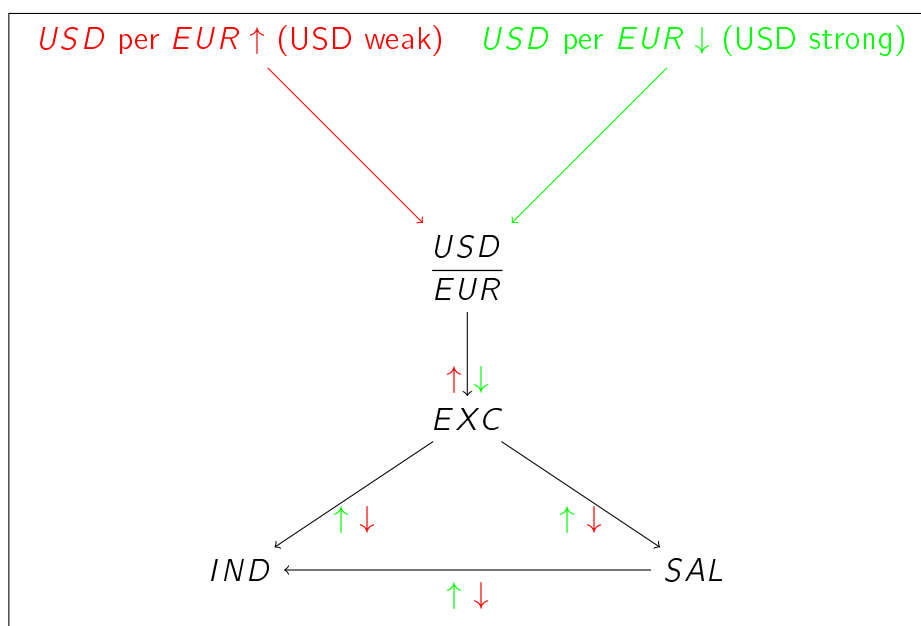


Figure 5.1: Relations between investigated variables

Source: Authors' design

5.2 The Data

We use monthly data from January 1999 to September 2009, therefore $T = 129$. The financial data are provided by Thomson Banker One. Prices of the shares traded at Frankfurt Stock Exchange are in Euro and are provided as monthly average closings. Each company is represented by its most traded share, in detail Porsche Automobil Holding Vz, BMW St, and Daimler St. BMW and Daimler shares are common shares. The Porsche share is a preferred share of Porsche Automobil Holding SE, which includes 100% of the Dr.Ing.h.c. F.Porsche AG. Common stocks of the Porsche Automobil Holding SE are not traded on stock markets.

The German premium car manufacturer stock price index (PCI) is calculated as a Laspeyres index by the following formula (see Moosmüller (2004) p.28)

$$PCI_t = \frac{w_{p0} \cdot p_t + w_{b0} \cdot b_t + w_{d0} \cdot d_t}{w_{p0} \cdot p_0 + w_{b0} \cdot b_0 + w_{d0} \cdot d_0}. \quad (5.2.1)$$

p_t indicates the Porsche share, b_t the BMW share, d_t the Daimler share and $w_{(\cdot)0}$ the weights belonging to the individual shares at time $t = 0$. As weights for a company we use its sales (in Euro, also provided by Thomson Banker One) divided by the sum of all companies sales in 1998. The weight factors are in detail $w_{b0} = 0.19$, $w_{d0} = 0.79$ and $w_{p0} = 0.02$. We use the logarithm of the series in the following. Figure 3.1 shows the PCI time series in logarithms. Logarithms have to be used to bring this series in accordance to the other ones concerning the behavior of the first differences. Although the RADF test indicates that the original series is stationary in first differences, huge outliers in the beginning of 2009 seem not to fit into the concept of a stationary series. Logarithms smooth out those outliers.

The monthly Dollar to Euro (Dollar per 1 Euro) exchange rate (EXC) is provided by the German Federal Reserve Bank (Bundesbank). It is plotted in figure 3.3, first differences in figure 3.4. The industrial production of the German car manufacturing and car parts manufacturing industry adjusted for working days (IND) is also provided by the Bundesbank. As it is originally scaled differently than the other values, a simple index is constructed with the average of 1999=100. The series is seasonally adjusted using the R-package `timsac`. We assume an additive model and split off the seasonal components, such that the series only consists of the AR-part and the noise. The US

Bureau of Economic Analysis provides the data on sales of foreign cars in the USA (SAL). Originally the scale of the series is sold units in thousands. Hence to bring it in accordance with the other data, an index with the average of 1999=100 is constructed. The series is seasonal adjusted like the industry production series.

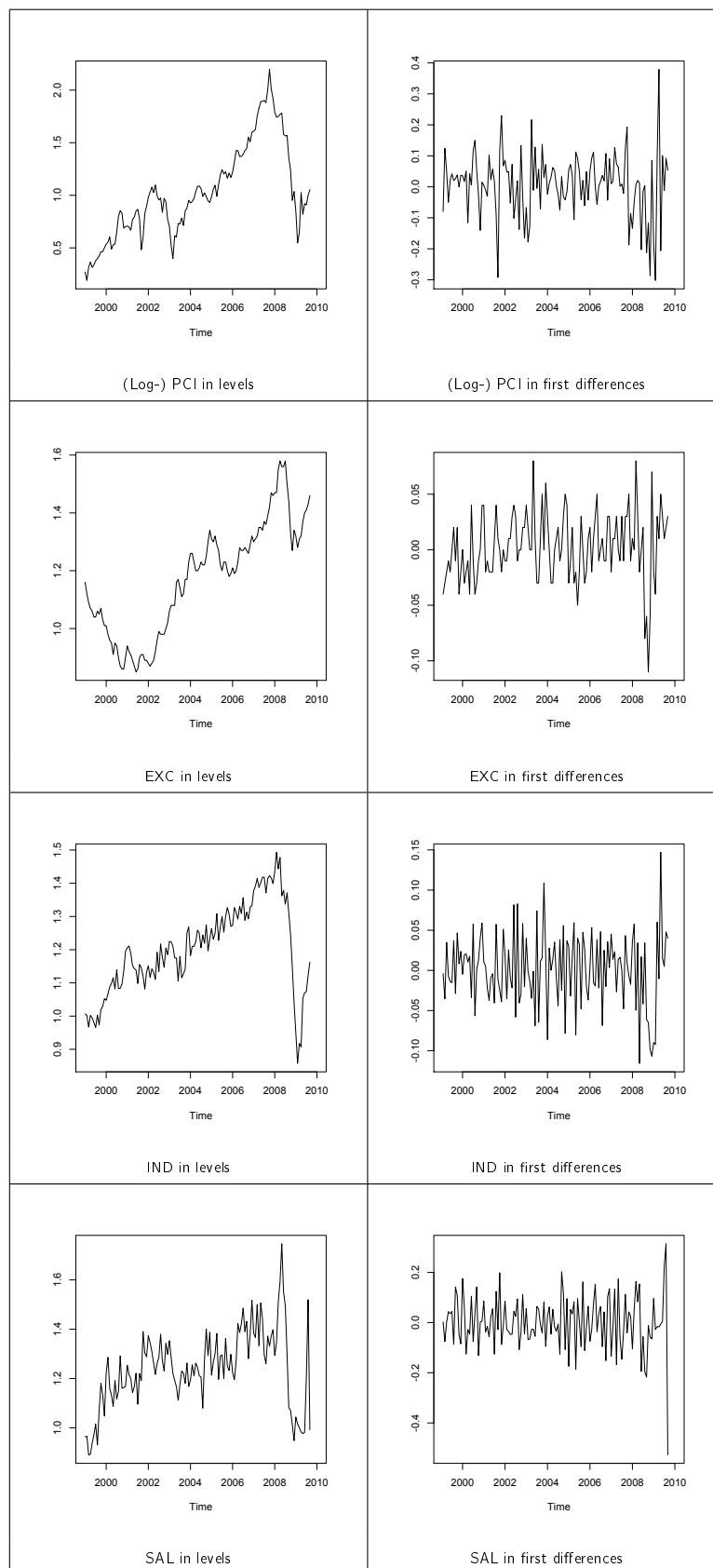


Figure 5.2: Data plot

Source: Authors' design

5.3 Nonlinearity and Stationarity Tests

Before AR-NN models can be adjusted the data have to be tested for stationarity. This test requires a specification of the lag-order, thus the test statistic is calculated for lag orders from $n = 1, \dots, 4$. We assume that a maximal lag order of $m = 4$ is sufficient, with respect to keep the number of parameters in a realistic relation to the number of observations T . In table 3.3 the RADF test statistic is calculated for levels and first differences for all series for lag orders 1 to 4 without constant and trend variables in the linear model (compare theorem 2.3). The function `rank()` in the basic package was used to compute the ranks of the series, the function `adfTest()` in the R-package `fUnitRoots` was used for calculation of the test statistics, critical values are taken from table 2.3 for a series with 100 observations. We clearly see that all series are nonstationary in levels and stationary in first differences (in other words integrated of order one).

But before AR-NN models are adjusted at the first differences of the series, their nonlinearity has to be examined. Therefore the Teräsvirta-Lin-Granger test (see section 3.1.2.2) with polynomial (3.1.20) is used. The R-package `tseries` provides a function `terasvirta.test()`, which is used to calculate the results. They are shown in tables 5.2 and 5.3. Note that here $T=128$ because of the differentiation. It is observable that all series are nonlinear at least for lag orders >1 .

As already mentioned above the data contain the crisis of 2008/2009. We now want to examine, if the crisis is responsible for the nonlinearity. Thus we test parts of the series without crisis (the first 100 values representing the time period from January 1999 to April 2007) for nonlinearity. The χ^2 - test statistic for the Lee-Teräsvirta-Granger test is shown in table 5.4. In fact it is observable, that series without crisis data tend to be linear. This result compared to the result above (table 5.2) outlines the nonlinear character of crises.

| Series | Lag | Levels | Differences | CV (5%) |
|--------|-----|---------|-------------|---------|
| PCI | 1 | -0.3733 | -8.1096 | -1.92 |
| | 2 | -0.3714 | -6.7666 | -1.92 |
| | 3 | -0.3285 | -5.0388 | -1.92 |
| | 4 | -0.4135 | -5.1162 | -1.92 |
| EXC | 1 | 0.2779 | -8.4932 | -1.92 |
| | 2 | 0.5457 | -6.4931 | -1.92 |
| | 3 | 0.5455 | -5.7527 | -1.92 |
| | 4 | 0.6129 | -4.7600 | -1.92 |
| IND | 1 | -0.6298 | -7.7757 | -1.92 |
| | 2 | -0.6926 | -6.0291 | -1.92 |
| | 3 | -0.7198 | -5.2267 | -1.92 |
| | 4 | -0.7122 | -4.5226 | -1.92 |
| SAL | 1 | -1.4447 | -11.3016 | -1.92 |
| | 2 | -1.0810 | -8.1241 | -1.92 |
| | 3 | -1.0387 | -7.2868 | -1.92 |
| | 4 | -0.9093 | -6.1205 | -1.92 |

Table 5.1: ADF test

| Series | Lag | χ^2 | df | CV (95%) |
|--------|-----|----------|----|----------|
| PCI | 1 | 11.5989 | 2 | 5.9915 |
| | 2 | 20.2478 | 7 | 14.0671 |
| | 3 | 44.2536 | 16 | 26.2962 |
| | 4 | 77.1989 | 30 | 43.773 |
| EXC | 1 | 4.7863 | 2 | 5.9915 |
| | 2 | 17.4252 | 7 | 14.0671 |
| | 3 | 39.6585 | 16 | 26.2962 |
| | 4 | 58.0129 | 20 | 43.773 |
| IND | 1 | 12.5899 | 2 | 5.9915 |
| | 2 | 18.9509 | 7 | 14.0671 |
| | 3 | 29.1732 | 16 | 26.2962 |
| | 4 | 76.7795 | 20 | 43.773 |
| SAL | 1 | 6.2537 | 2 | 5.9915 |
| | 2 | 22.5614 | 7 | 14.0671 |
| | 3 | 33.1896 | 16 | 26.2962 |
| | 4 | 49.5517 | 20 | 43.773 |

Table 5.2: Teräsvirta-Lin-Granger test χ^2 - statistic (data in first differences)

| Series | Lag | F | df | CV (95%) |
|--------|-----|--------|--------|----------|
| PCI | 1 | 5.9281 | 2;125 | 3.0687 |
| | 2 | 2.9135 | 7;119 | 2.0874 |
| | 3 | 2.8137 | 16;109 | 1.7371 |
| | 4 | 2.5938 | 30;94 | 1.5806 |
| EXC | 1 | 2.3813 | 2;125 | 3.0687 |
| | 2 | 2.4792 | 7;119 | 2.0874 |
| | 3 | 2.4743 | 16;109 | 1.7371 |
| | 4 | 1.7966 | 30;94 | 1.5806 |
| IND | 1 | 6.4599 | 2;125 | 3.0687 |
| | 2 | 2.7128 | 7;119 | 2.0874 |
| | 3 | 1.7439 | 16;109 | 1.7371 |
| | 4 | 2.5751 | 30;94 | 1.5806 |
| SAL | 1 | 3.1294 | 2;125 | 3.0687 |
| | 2 | 3.2767 | 7;119 | 2.0874 |
| | 3 | 2.0166 | 16;109 | 1.7371 |
| | 4 | 1.4812 | 30;94 | 1.5806 |

Table 5.3: Teräsvirta-Lin-Granger test F - statistic (data in first differences)

| Series | Lag | χ^2 | df | CV (95%) |
|--------|-----|----------|----|----------|
| PCI | 1 | 2.3462 | 2 | 5.9915 |
| | 2 | 11.199 | 7 | 14.0671 |
| | 3 | 24.3844 | 16 | 26.2962 |
| | 4 | 51.8294 | 30 | 43.773 |
| EXC | 1 | 1.9647 | 2 | 5.9915 |
| | 2 | 7.3563 | 7 | 14.0671 |
| | 3 | 24.7121 | 16 | 26.2962 |
| | 4 | 50.1251 | 20 | 43.773 |
| IND | 1 | 0.4235 | 2 | 5.9915 |
| | 2 | 8.8055 | 7 | 14.0671 |
| | 3 | 24.2016 | 16 | 26.2962 |
| | 4 | 58.0233 | 20 | 43.773 |
| SAL | 1 | 2.4662 | 2 | 5.9915 |
| | 2 | 8.4997 | 7 | 14.0671 |
| | 3 | 14.2617 | 16 | 26.2962 |
| | 4 | 42.6048 | 20 | 43.773 |

Table 5.4: Teräsvirta-Lin-Granger test χ^2 - statistic without crisis data (data in first differences, first 100 values)

5.4 Univariate AR-NN

5.4.1 Lag Selection

We use the series in first differences as a starting point for lag selection, because the lag selection procedures in section 3.2 are only appropriate for stationary series. Figure 5.3 shows the AC and PAC for all series up to lag order 10. In the figure the interval $[-\frac{2}{\sqrt{T}}; \frac{2}{\sqrt{T}}] = [-0.177; 0.177]$ is marked by a dashed line. The PAC is above that dashed line for the IND series at lag 1 and for the SAL series at lag 2. This information can be used for lag selection, but we still have no indication for the lag order for the first two series.

Hence we subsequently apply the other lag selection procedures proposed in section 3.2. The MIC is - like the AC - not a very useful tool for lag selection. The MIC, calculated by the function in appendix B.3 with $d = 100$, for the first 4 lags are shown in table 5.5. It is observable that the results are in the same range as the $|AC|$. Results for polynomial lag selection using formula (3.2.17) and the AIC and the BIC as quality criteria are displayed in table 5.6. For the calculations the function in appendix B.2 is used, the optimal lag orders are marked by a "*". AIC and BIC accord in only one case. The AIC sometimes also tends to include more lags than the BIC. The next table (table 5.7) shows results for the Nonlinear Final Prediction Error (NFPE) from section 3.2.4. For calculation of the results the software JMulti was used. The AFPE as well as its correction, the CAFPE, are calculated. Both criteria have to be minimized like any other IC. It is observable that both criterions indicate the same optimal lag order.

As the different lag selection procedures lead to partly very different results, we have to decide which lag order we use in the following. We proceed with the results from polynomial lag selection and use the AIC, thus the PCI has 4 lags, the EXC series has 3 lags and the IND and the SAL series have 2 lags.

| Series | Lags | | | |
|--------|--------|--------|--------|--------|
| | 1 | 2 | 3 | 4 |
| PCI | 0.0164 | 0.0312 | 0.0448 | 0.0575 |
| EXC | 0.0675 | 0.0979 | 0.1208 | 0.1399 |
| IND | 0.0243 | 0.0448 | 0.0627 | 0.0788 |
| SAL | 0.0116 | 0.0225 | 0.0328 | 0.0426 |

Table 5.5: MIC

| Series | Lag | AIC | BIC |
|--------|-----|------------|------------|
| PCI | 1 | -215.2964 | -218.0006* |
| | 2 | -213.1339 | -213.4044 |
| | 3 | -209.6199 | -203.8683 |
| | 4 | -216.0644* | -195.3435 |
| EXC | 1 | -538.8916 | -541.5958* |
| | 2 | -535.8858 | -536.1562 |
| | 3 | -541.6615* | -535.9100 |
| | 4 | -538.0680 | -517.3471 |
| IND | 1 | -428.1411 | -430.8452 |
| | 2 | -435.8922* | -436.1626* |
| | 3 | -428.1819 | -422.4304 |
| | 4 | -430.2283 | -409.5074 |
| SAL | 1 | -210.4986 | -213.2028 |
| | 2 | -224.1334* | -224.4039* |
| | 3 | -220.1187 | -214.3672 |
| | 4 | -214.0852 | -193.3643 |

Table 5.6: Polynomial approximation lag selection

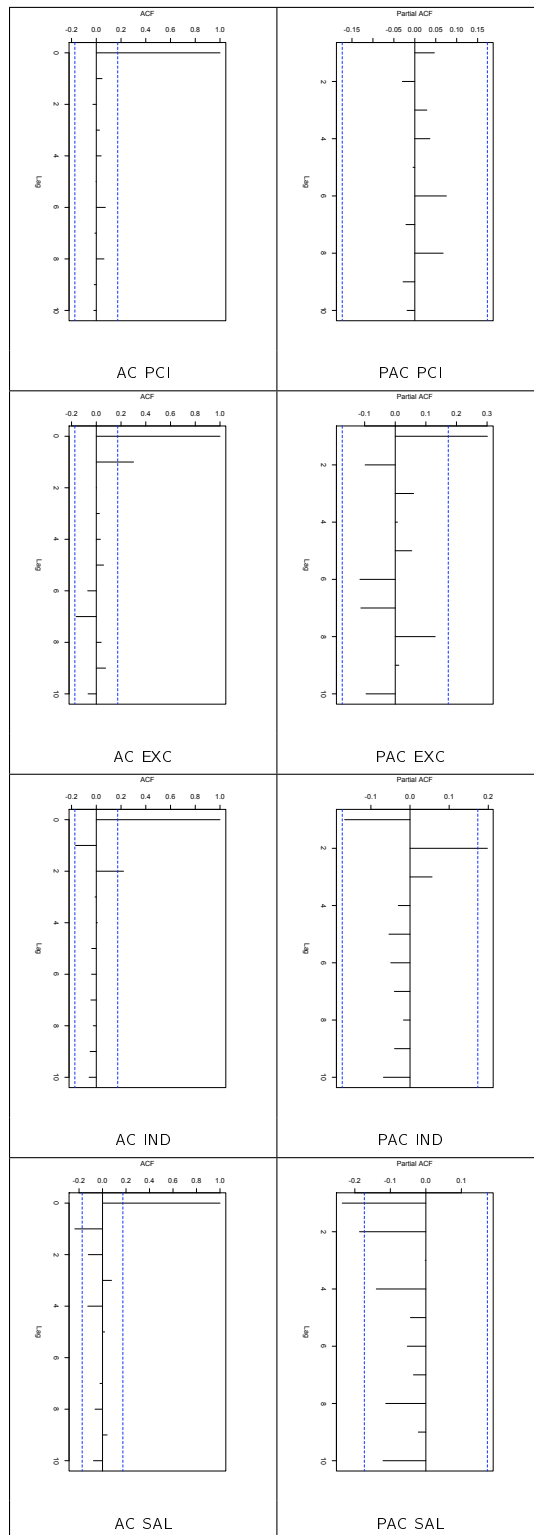


Figure 5.3: AC and PAC

Source: Authors' design

| Series | Lag | AFPE | CAFPE |
|---------------|------------|-------------|--------------|
| PCI | 1 | 0.0099* | 0.0101* |
| | 2 | 0.0103 | 0.0111 |
| | 3 | 0.0287 | 0.0342 |
| | 4 | 0.0409 | 0.0556 |
| EXC | 1 | 0.0008 | 0.0008 |
| | 2 | 0.0007* | 0.0008* |
| | 3 | 0.0008 | 0.0010 |
| | 4 | 0.0011 | 0.0015 |
| IND | 1 | 0.0019* | 0.0019* |
| | 2 | 0.0020 | 0.0021 |
| | 3 | 0.0023 | 0.0028 |
| | 4 | 0.0029 | 0.0040 |
| SAL | 1 | 0.0095 | 0.0097 |
| | 2 | 0.0086* | 0.0093* |
| | 3 | 0.0094 | 0.0112 |
| | 4 | 0.0104 | 0.0141 |

Table 5.7: NFPE

5.4.2 Estimation and Bottom-Up Parameter Tests

In the following the first 120 values of the sample are used for estimation of the models (training set (TS), $T=120$), the values 121 to 128 for comparison with out-of-sample predictions (test subset). One- and eight-step predictions are compared to various other linear and nonlinear models. For estimation of the parameters some initial settings have to be defined. The relationship ES/TS is 0.98 and thus the VS contains 2 values. $\lambda = 0.001$ and $\tau = 100$ seems to be the best setting for those parameters. The initial values for the linear part are estimated by OLS, the initial values for the nonlinear part are set 1 uniformly. The Levenberg-Marquardt algorithm has 5 iterations to initialize. This lowers the dependency of the results from the initial parameter values. One has to be cautious with setting such a limit for initialization: Empirical application shows, if that limit is too high, the residuals tend to be no more independent for some series. Therefore it can be said that too much iterations for initialization of the algorithm may lead to misspecified models.

Subsequently the stopped training concept is applied: The optimal model is reached if the VS-RSS is minimal. A maximal number of iterations of 200 is used (thus 195 iterations are used for stopped training, the first five iterations are needed to initialize the values, see section 3.3.7). Table 5.8 shows the optimal number of iterations (i^*) for all series for $h=0,1,2,3,4$.

In tables 5.10 to 5.13 estimation and prediction results are shown for AR-NN with $h=0,1,2,3,4$ (calculated using the functions `estimate.ARNN()`, `fitted.ARNN()` and `residuals.ARNN()` in appendix B.4) and some alternative models. In detail those are various ARMA models (calculated using the function `arima()` in the `stats` package), the logistic smooth transition regression (LSTAR) model and a local linear kernel regression model (using the functions `lstar()` and `llar()` in the package `tsDyn`). The in-sample and the one- and eight step out-of-sample root mean-squared error (RMSE), calculated by

$$\text{RMSE} = \sqrt{\frac{1}{T} \sum_{t=1}^T (x_t - \hat{x}_t)^2} \quad (5.4.1)$$

for in-sample and

$$\text{RMSE} = \sqrt{\frac{1}{k} \sum_{t=T+1}^k (x_t - \hat{x}_t)^2} \quad (5.4.2)$$

for the k -step prediction are shown in the tables as performance measure. The in-sample RMSE of the models trained by cross-validation stopping can not be compared to the RMSE of the alternative models. They only minimize the RSS of the TS, whereas the cross-validation stopping method also considers the prediction behavior during the estimation process (VS as part of the TS). A fairer comparison is the application of the ES-RMSE as in-sample measure for the AR-NN's. It is minimized analogously to the in-sample RMSE in the alternative models. Note that different variances for the individual steps in more-step prediction might distort the RMSE. A rolling one-step prediction would overcome that problem. However more-step prediction is a realistic aim in forecasting time series, thus we apply it despite that. The figure 5.4 should be considered in together with the RMSE for more-step prediction. In addition tables 5.10 to 5.13 show the Theil inequality coefficient (IEC) for the out-of-sample set. The Theil IEC for a k -step prediction is calculated by

$$\text{Theil IEC} = \sqrt{\frac{\sum_{t=T+1}^k (x_t - \hat{x}_t)^2}{\sum_{t=T+1}^k (x_t - \frac{1}{T} \sum_{t=1}^T x_t)^2}}. \quad (5.4.3)$$

It compares the out-of-sample prediction with the mean of the TS. If the Theil IEC is small, the out-of-sample prediction is better, if it is larger than one, the mean of the TS (average of the TS) is better (see Steurer (1996) p.120). The Theil IEC also simplifies the comparison of the out-of-sample performance of the models between each other, because it normalizes the RMSE (by the division of the out-of-sample MSE by the mean of the TS MSE).

To identify the optimal model we use the two bottom-up procedures proposed in section 3.4.1:

- Table 5.9 shows the Lee-White-Granger bottom up test for $h=1$ using the models estimated in tables 5.10 to 5.13 (calculated using the function `LWG.test()` in appendix B.9). We only use the first 118 values to calculate the test statistic. This is necessary because the Lee-White-Granger test is based on the in-sample performance for RSS minimizing procedures. If models are estimated using the

stopped training method, only the ES is considered to be in-sample and is minimized like in least squares procedures. Including the VS would distort the results of the test. We execute the test for the model with $h=1$ (the test for $h=0$ equals the linearity test which was already executed in section 5.3). The test indicates, that for none of the series a second hidden neuron would improve the model. The statement of the test is correct if the third column in tables 5.10 to 5.13 are considered. If the focus is mainly on in-sample behavior of the models, the estimation procedure could be stopped after the models with $h=1$ are estimated.

- As we are also interested in the out-of-sample performance, cross validation considering the one- or eight- step- prediction would be an alternative. We assume that the models with the lowest out-of-sample RMSE are optimal. Concerning the one step prediction those are the AR-NN with $h=4$ for PCI and EXC, $h=1$ for IND and $h=0$ for SAL. If the eight-step prediction is considered the optimal models have $h=4$ for PCI, EXC and IND and again $h=0$ for SAL. Consequently the SAL series is linear (in contrast to the result of the nonlinearity test).

We proceed with the analysis of the models which perform best concerning the one-step prediction. In figure 5.4 the in-sample behavior is plotted, figure 5.5 shows the out-of-sample performance compared to a linear AR and a ARMA model. As the AR-NN for SAL has $h=0$, it does not differ from the linear AR model. In particular in the short-run prediction (1-3 steps) the AR-NN performs better than linear models, whereas for higher steps of prediction they become similar to the linear AR.

Tables 5.14 to 5.17 show the parameters for the mentioned models. Interpretation of the parameters is certainly more difficult than in linear models - if not impossible. One argument might be that the larger the parameter for a variable in one hidden neuron is, the more this variable contributes to the nonlinearity of the hidden neuron. Note that for the models with $h=4$ the parameters in all hidden neurons are the same. If only one hidden neuron with $\beta = 4 \cdot \beta_j$ would be used instead, the results would be identical. This could be one way to reduce the complexity.²

²This result depends from the initial settings. For different settings, the parameters in each hidden neuron differ from each other.

| h | PCI | EXC | IND | SAL |
|----------|------------|------------|------------|------------|
| 0 | 5 | 5 | 138 | 5 |
| 1 | 5 | 5 | 200 | 94 |
| 2 | 5 | 5 | 200 | 5 |
| 3 | 5 | 5 | 200 | 5 |
| 4 | 5 | 5 | 200 | 5 |

Table 5.8: Iterations necessary for univariate models

The nonlinearity can be shown by so-called surface plots (see figures 5.6 to 5.9). Therefore only the first two lags are considered and plotted against the estimated values, for values between -1 to 1 for each lag. Other lags are kept constant. Surface plots for models with $h=0,1,2,3,4$ from tables 5.10 to 5.13 are shown.

| Series | χ^2 | df | CV (95%) | F | df | CV (95%) |
|---------------|----------|-----------|-----------------|----------|-----------|-----------------|
| PCI | 3.4221 | 30 | 43.7730 | 0.1895 | 30;83 | 1.5966 |
| EXC | 0.02081 | 16 | 26.2962 | 1.5088 | 16;98 | 1.7477 |
| IND | 5.1319 | 7 | 14.0671 | 0.9338 | 7;108 | 2.0955 |
| SAL | 0.4482 | 7 | 14.0671 | 1.4122 | 7;108 | 2.0955 |

Table 5.9: Lee-White-Granger test for $h=1$

| Model | r | In-sample | | Out-of-sample 1 step prediction | | Out-of-sample 8 step prediction | |
|---------------------|----|-----------|-----------|---------------------------------|-----------|---------------------------------|-----------|
| | | RMSE | Theil IEC | RMSE | Theil IEC | RMSE | Theil IEC |
| AR-NN's | | | | | | | |
| AR-NN(4), h=0 | 5 | 0.0900 | 1.005 | 0.3084 | 1.005 | 0.1957 | 0.9991 |
| AR-NN(4), h=1 | 11 | 0.0898 | 0.9819 | 0.3013 | 0.9819 | 0.1954 | 0.9974 |
| AR-NN(4), h=2 | 17 | 0.0897 | 0.8791 | 0.2698 | 0.8791 | 0.1905 | 0.9722 |
| AR-NN(4), h=3 | 23 | 0.0898 | 0.8583 | 0.2634 | 0.8583 | 0.1892 | 0.9658 |
| AR-NN(4), h=4 | 29 | 0.0900 | 0.8553 | 0.2624 | 0.8553 | 0.1889 | 0.9640 |
| Alternatives | | | | | | | |
| AR(4) | 5 | 0.0915 | 1.0397 | 0.3191 | 1.0397 | 0.1969 | 1.0051 |
| ARMA(4,1) | 6 | 0.0915 | 1.0363 | 0.3180 | 1.0363 | 0.1985 | 1.0132 |
| ARMA(4,5) | 10 | 0.0844 | 1.0617 | 0.3258 | 1.0617 | 0.2221 | 1.1337 |
| LSTAR(4) | 10 | 0.0869 | 1.1083 | 0.3401 | 1.1083 | 0.2043 | 1.0429 |
| LLAR(4) | - | 0.0964 | 1.0565 | 0.3242 | 1.0565 | 0.1975 | 1.0081 |

Table 5.10: PCI: AR-NN vs. other models

| Model | r | In-sample | | Out-of-sample 1 step prediction | | Out-of-sample 8 step prediction | |
|---------------------|----|-----------|-----------|---------------------------------|-----------|---------------------------------|-----------|
| | | RMSE | Theil IEC | RMSE | Theil IEC | RMSE | Theil IEC |
| AR-NN's | | | | | | | |
| AR-NN(3), h=0 | 4 | 0.0276 | 0.6871 | 0.0284 | 0.6871 | 0.0287 | 0.9674 |
| AR-NN(3), h=1 | 9 | 0.0270 | 0.7597 | 0.0314 | 0.7597 | 0.0272 | 0.9169 |
| AR-NN(3), h=2 | 14 | 0.0271 | 0.6581 | 0.0272 | 0.6581 | 0.0266 | 0.8966 |
| AR-NN(3), h=3 | 19 | 0.0271 | 0.5710 | 0.0236 | 0.5710 | 0.0262 | 0.8831 |
| AR-NN(3), h=4 | 24 | 0.0272 | 0.5032 | 0.0208 | 0.5032 | 0.0260 | 0.8764 |
| Alternatives | | | | | | | |
| AR(3) | 4 | 0.0288 | 0.6242 | 0.0258 | 0.6242 | 0.0277 | 0.9337 |
| ARMA(3,1) | 5 | 0.0288 | 0.6387 | 0.0264 | 0.6387 | 0.0277 | 0.9337 |
| ARMA(3,5) | 9 | 0.0272 | 0.9073 | 0.0375 | 0.9073 | 0.0292 | 0.9843 |
| LSTAR(3) | 9 | 0.0263 | 1.9766 | 0.0817 | 1.9766 | 0.0255 | 0.8596 |
| LLAR(3) | - | 0.0303 | 2.0056 | 0.0829 | 2.0056 | 0.0270 | 0.9101 |

Table 5.11: EXC: AR-NN vs. other models

| Model | r | In-sample | | Out-of-sample 1 step prediction | | Out-of-sample 8 step prediction | |
|---------------------|----|-----------|-----------|---------------------------------|-----------|---------------------------------|-----------|
| | | RMSE | Theil IEC | RMSE | Theil IEC | RMSE | Theil IEC |
| AR-NN's | | | | | | | |
| AR-NN(2), h=0 | 3 | 0.0388 | 1.3913 | 0.1274 | 1.3913 | 0.0765 | 1.1075 |
| AR-NN(2), h=1 | 7 | 0.0378 | 0.1332 | 0.0122 | 0.1332 | 0.0690 | 0.9990 |
| AR-NN(2), h=2 | 11 | 0.0378 | 0.2632 | 0.0241 | 0.2632 | 0.0652 | 0.9439 |
| AR-NN(2), h=3 | 15 | 0.0378 | 0.3451 | 0.0316 | 0.3451 | 0.0642 | 0.9295 |
| AR-NN(2), h=4 | 19 | 0.0378 | 0.4084 | 0.0374 | 0.4084 | 0.0639 | 0.9251 |
| Alternatives | | | | | | | |
| AR(2) | 3 | 0.0415 | 1.0004 | 0.0916 | 1.0004 | 0.0715 | 1.0352 |
| ARMA(2,1) | 4 | 0.0410 | 0.6727 | 0.0616 | 0.6727 | 0.0903 | 1.3073 |
| ARMA(2,5) | 8 | 0.0399 | 0.5559 | 0.0509 | 0.5559 | 0.0973 | 1.4087 |
| LSTAR(2) | 8 | 0.0378 | 2.9094 | 0.2664 | 2.9094 | 0.1424 | 2.0616 |
| LLAR(2) | - | 0.0445 | 1.4831 | 0.1358 | 1.4831 | 0.0748 | 1.0829 |

Table 5.12: IND: AR-NN vs. other models

| Model | r | In-sample | | Out-of-sample 1 step prediction | | Out-of-sample 8 step prediction | |
|-----------------------|----|-----------|-----------|---------------------------------|-----------|---------------------------------|-----------|
| | | RMSE | Theil IEC | RMSE | Theil IEC | RMSE | Theil IEC |
| AR-NN's | | | | | | | |
| AR-NN(2), h=0 | 3 | 0.0905 | 0.0104 | 0.3664 | 0.2311 | 0.999 | 0.999 |
| AR-NN(2), h=1 | 7 | 0.0879 | 0.0223 | 0.7855 | 0.2316 | 1.0012 | 1.0012 |
| AR-NN(2), h=2 | 11 | 0.0881 | 0.0178 | 0.627 | 0.2312 | 0.9994 | 0.9994 |
| AR-NN(2), h=3 | 15 | 0.0883 | 0.0168 | 0.5918 | 0.2312 | 0.9994 | 0.9994 |
| AR-NN(2), h=4 | 19 | 0.0884 | 0.0164 | 0.5777 | 0.2312 | 0.9994 | 0.9994 |
| Alternatives | | | | | | | |
| AR(2) | 3 | 0.0903 | 0.0107 | 0.3769 | 0.2311 | 0.999 | 0.999 |
| ARMA(2,1) | 4 | 0.0896 | 0.007 | 0.2466 | 0.2315 | 1.0007 | 1.0007 |
| ARMA(2,5) | 8 | 0.0860 | 0.1019 | 3.5896 | 0.2360 | 1.0202 | 1.0202 |
| LSTAR(2) ^a | - | - | - | - | - | - | - |
| LLAR(2) | - | 0.0942 | 0.0107 | 0.3769 | 0.2311 | 0.999 | 0.999 |

Table 5.13: SAL: AR-NN vs. other models

^aNote that the LSTAR model was not calculated, because the R function indicated, that the series is linear

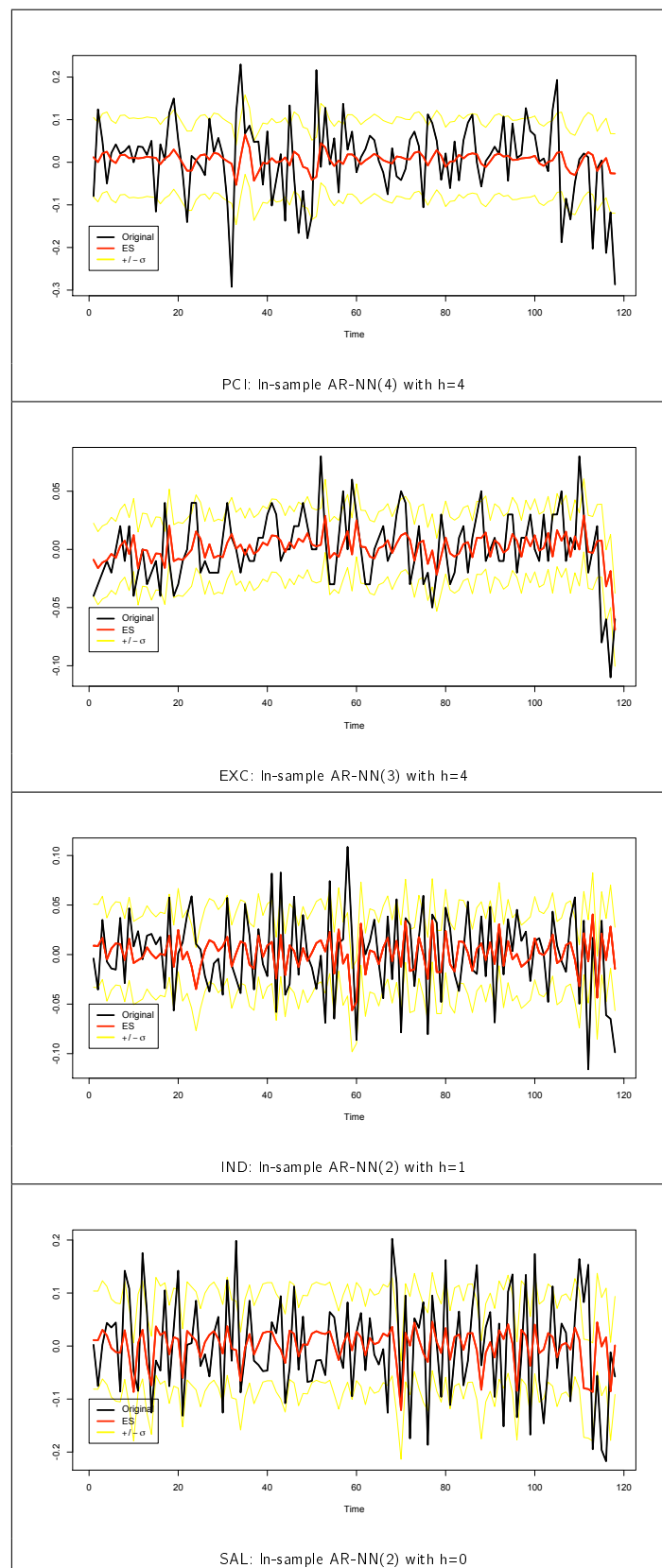


Figure 5.4: Univariate models in-sample plots

Source: Authors' design

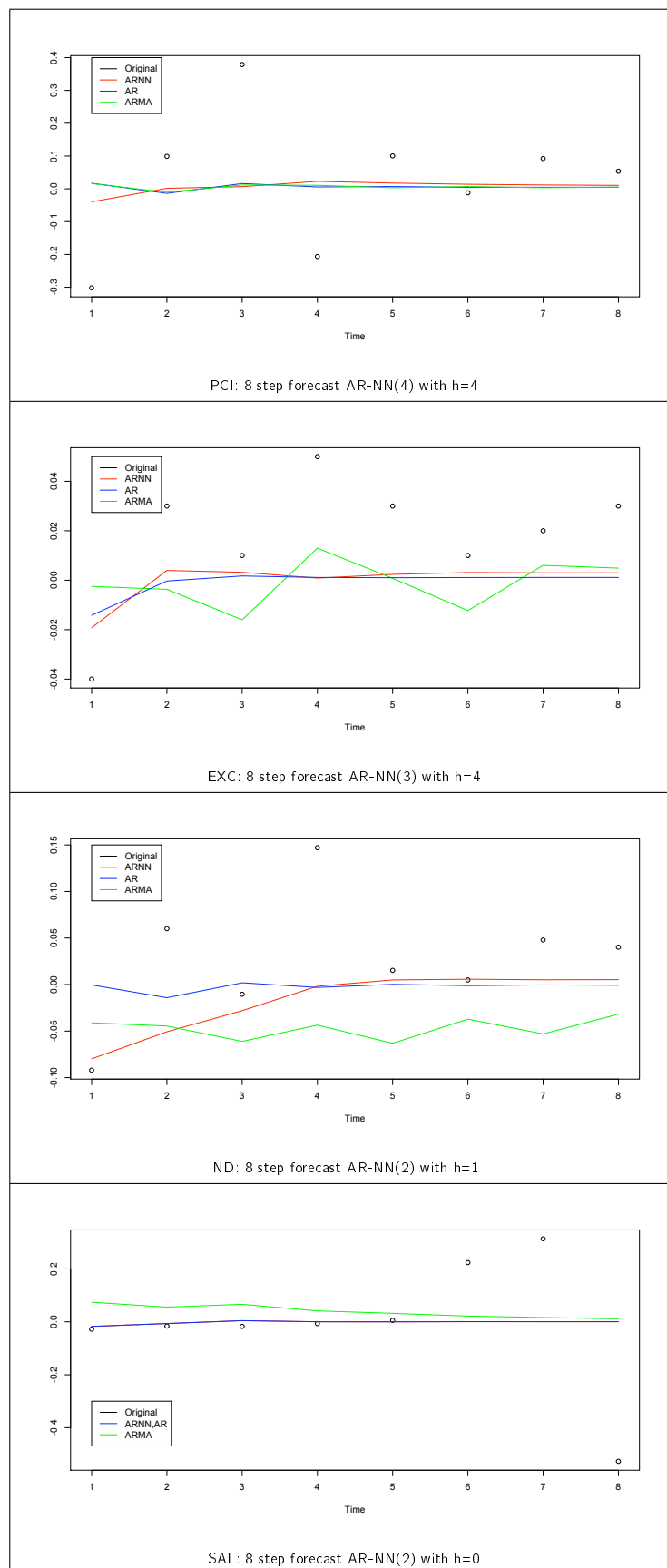


Figure 5.5: Univariate models out-of-sample plots

Source: Authors' design

| | | |
|------------------------|------------------------|------------------------|
| $\alpha_0 = -0.3247$ | $\gamma_{01} = 0.6347$ | $\gamma_{02} = 0.6347$ |
| $\alpha_1 = -0.2901$ | $\gamma_{11} = 1.0051$ | $\gamma_{12} = 1.0051$ |
| $\alpha_2 = -0.2620$ | $\gamma_{21} = 0.7245$ | $\gamma_{22} = 0.7245$ |
| $\alpha_3 = -0.5168$ | $\gamma_{31} = 1.1304$ | $\gamma_{32} = 1.1304$ |
| $\alpha_4 = -0.3209$ | $\gamma_{41} = 0.6106$ | $\gamma_{42} = 0.6106$ |
| | $\beta_1 = 0.1499$ | $\beta_2 = 0.1499$ |
| $\gamma_{03} = 0.6347$ | $\gamma_{04} = 0.6347$ | |
| $\gamma_{13} = 1.0051$ | $\gamma_{14} = 1.0051$ | |
| $\gamma_{23} = 0.7245$ | $\gamma_{24} = 0.7245$ | |
| $\gamma_{33} = 1.1304$ | $\gamma_{34} = 1.1304$ | |
| $\gamma_{34} = 0.6106$ | $\gamma_{44} = 0.6106$ | |
| $\beta_3 = 0.1499$ | $\beta_4 = 0.1499$ | |

Table 5.14: PCI: Parameters AR-NN(4) with h=4

| | | |
|------------------------|------------------------|------------------------|
| $\alpha_0 = -0.8535$ | $\gamma_{01} = 1.0348$ | $\gamma_{02} = 1.0348$ |
| $\alpha_1 = -0.1592$ | $\gamma_{11} = 1.2125$ | $\gamma_{12} = 1.2125$ |
| $\alpha_2 = -0.6814$ | $\gamma_{21} = 1.2985$ | $\gamma_{22} = 1.2985$ |
| $\alpha_3 = -0.2733$ | $\gamma_{31} = 0.8475$ | $\gamma_{32} = 0.8475$ |
| | $\beta_1 = 0.2757$ | $\beta_2 = 0.2757$ |
| $\gamma_{03} = 1.0348$ | $\gamma_{04} = 1.0348$ | |
| $\gamma_{13} = 1.2125$ | $\gamma_{14} = 1.2125$ | |
| $\gamma_{23} = 1.2985$ | $\gamma_{24} = 1.2985$ | |
| $\gamma_{33} = 0.8475$ | $\gamma_{34} = 0.8475$ | |
| $\beta_3 = 0.2757$ | $\beta_4 = 0.2757$ | |

Table 5.15: EXC: Parameters AR-NN(3) with h=4

| | |
|----------------------|------------------------|
| $\alpha_0 = -1.1246$ | $\gamma_{01} = 1.0516$ |
| $\alpha_1 = -1.4247$ | $\gamma_{11} = 1.9216$ |
| $\alpha_2 = -1.6643$ | $\gamma_{21} = 2.9492$ |
| | $\beta_1 = 1.4468$ |

Table 5.16: IND: Parameters AR-NN(2) with $h=1$

| |
|----------------------|
| $\alpha_0 = 0.0008$ |
| $\alpha_1 = -0.2710$ |
| $\alpha_2 = -0.1212$ |

Table 5.17: SAL: Parameters AR-NN(2) with $h=0$

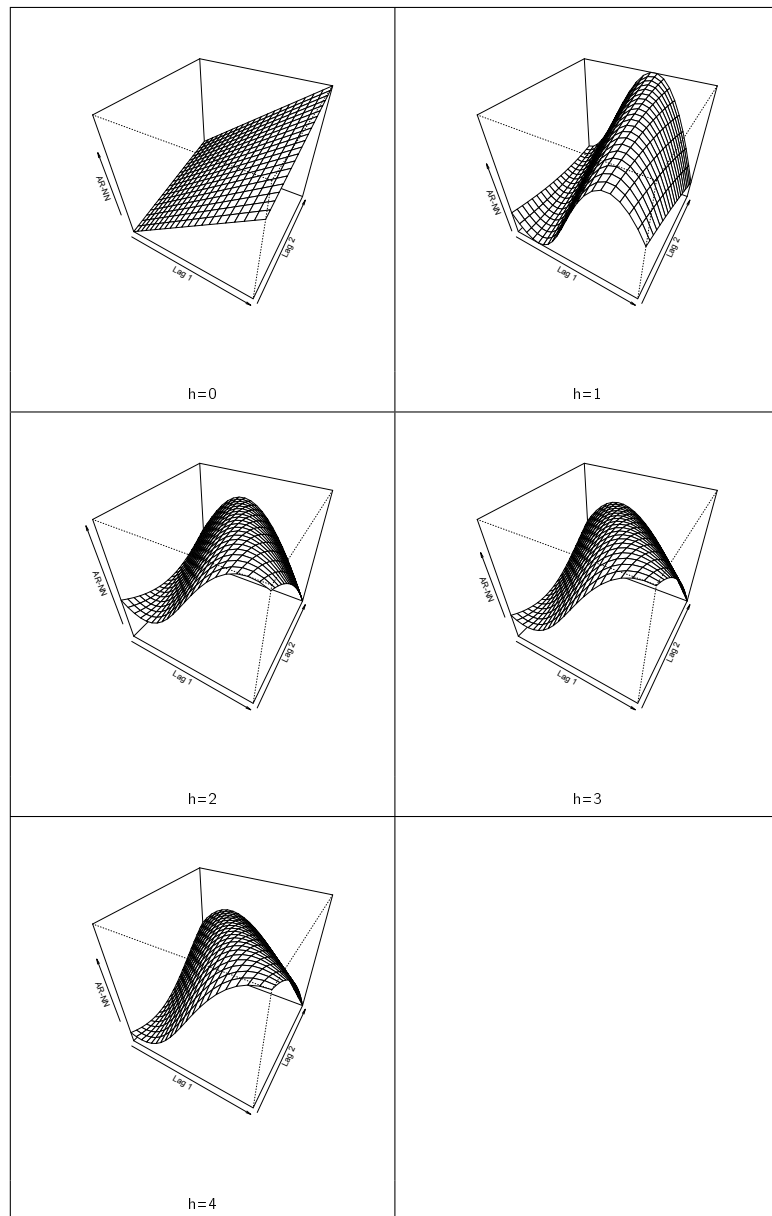


Figure 5.6: PCI: Surface plot AR-NN(4) with various h

Source: Authors' design

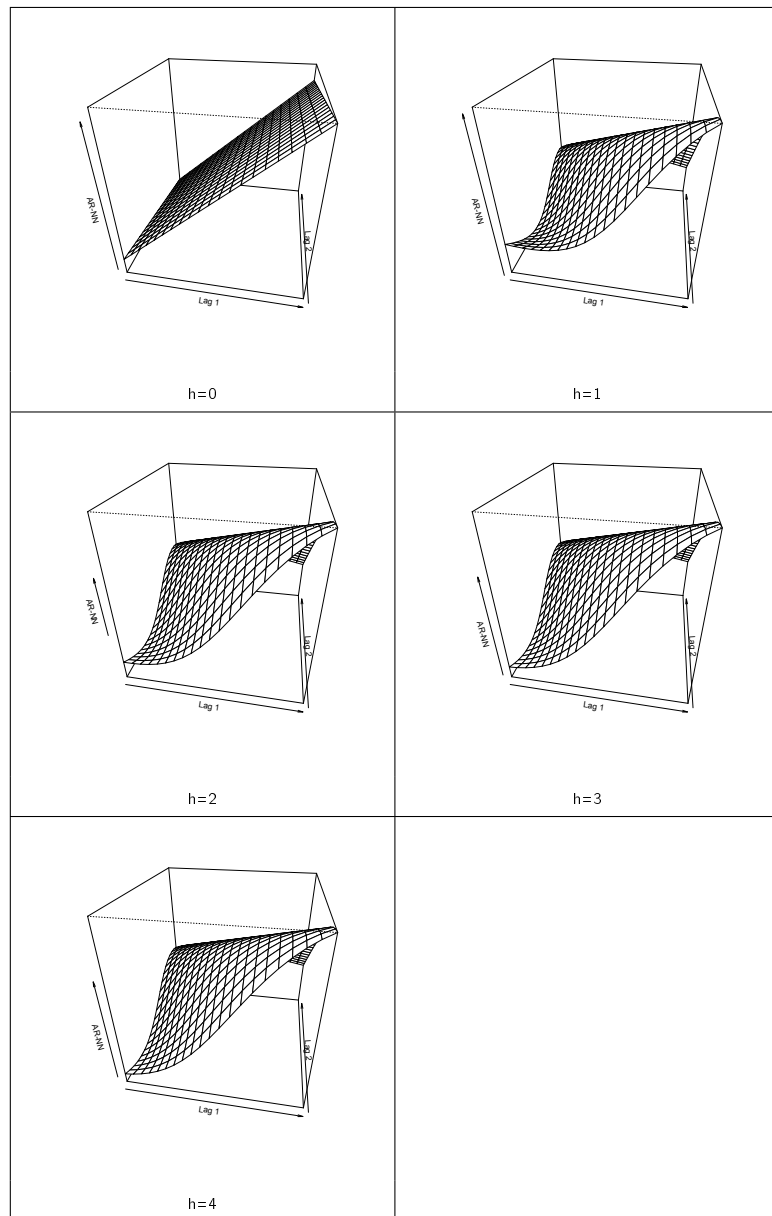


Figure 5.7: EXC: Surface plot AR-NN(3) with various h

Source: Authors' design

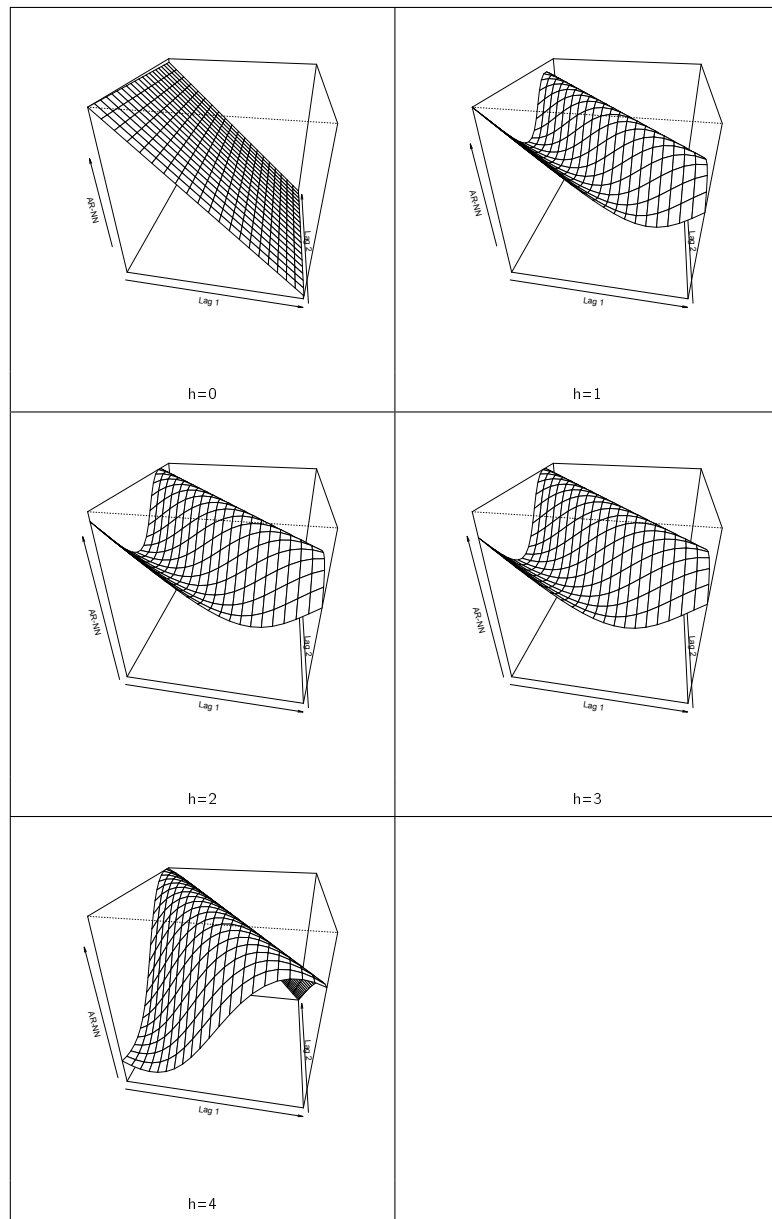


Figure 5.8: IND: Surface plot AR-NN(2) with various h

Source: Authors' design

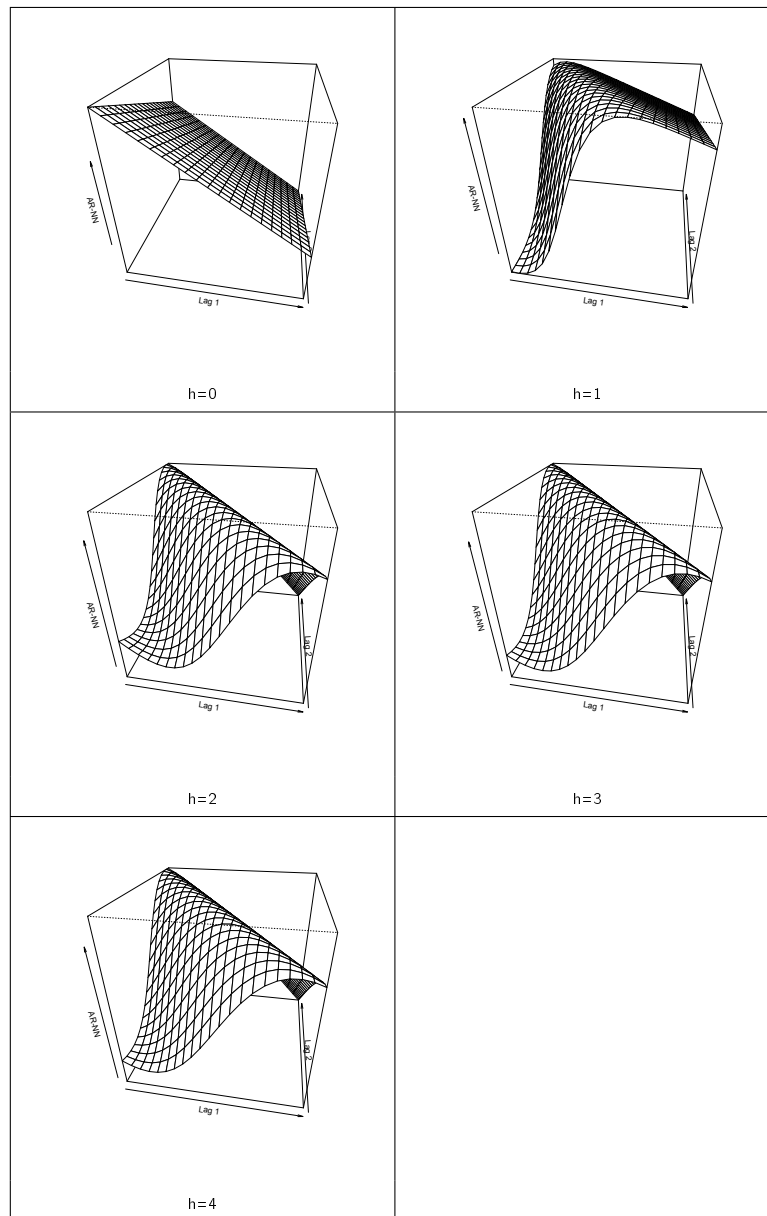


Figure 5.9: SAL: Surface plot AR-NN(2) with various h

Source: Authors' design

5.4.3 Top-Down Parameter Tests

Again we examine the models which perform best concerning the one step prediction. Condition 3.3 (see section 3.4.2.1) required for consistency is violated by the models for the PCI and EXC series with $h = 4$. For the other models the top-down parameter tests can be executed. Table 5.18 shows the NIC for the IND series. The model for the SAL series is already a linear, therefore a NIC (which tests the significance of nonlinear parts) is not calculated. We can also calculate the Wald test statistic for the α as well as the γ -parameters. Results are shown in table 5.19. All nonlinear hidden units and parameters seem to be significant. For the results -like above, and for the same reasons - only the ES values have been used. Results can be calculated using the function `covariance.ARNN()` in appendix B.8.

| Series | h=0 | h=1 |
|--------|--------|--------|
| IND | 1.3156 | 0.0014 |

Table 5.18: Univariate models: NIC

| Weight | Test statistic | |
|---------------|----------------|------------|
| | EXC | SAL |
| α_0 | 497190 | 2 |
| α_1 | 183330849 | 2056152361 |
| α_2 | 292475532 | 1214875 |
| γ_{01} | 575565 | |
| γ_{11} | 504128246 | |
| γ_{21} | 1259951220 | |

Table 5.19: Univariate models: Wald test

5.4.4 Residual Analysis

Finally the residuals of the estimated models are evaluated (again for the models which perform best concerning the one step prediction). Therefore we examine, if the residuals are in accordance with the i.i.d. $N(0, \sigma^2)$ assumption. Only the residuals for the ES subset are examined, because the in-sample RSS-minimization only applies to them.

At first we test the normality. Figure 5.10 shows the density histograms belonging to those residual series, including the density function of a normal distribution. The normality can be analyzed by looking at the histograms and the third and fourth moments (see table 5.20, results are calculated using the functions `skewness()` and `kurtosis()` in the package `e1071`). The skewness should be zero and the kurtosis around 3 if the residuals are Gaussian distributed. In addition the Jarque-Bera test is executed to examine the normality of the residuals. Except for the PCI series all residuals are normal distributed according to this test (see table 5.21).

The next part of the assumption is the independence. Therefore we consider the AC of all residual series (see figure 5.11). In addition we calculate the Box-Pierce statistic for lag orders between 1 and 5 (using the function `Box.test()` in the package `tseries`). The test statistic is χ^2 -distributed with degrees of freedom equal to the number of lags. All series seem to be independent according to the AC, as it is not significant at any lag. According to the Box-Pierce statistic the residuals of all series are independent.

Finally heteroskedasticity can be tested by the ARCH-LM test of Engle (1982) (ARCH means Autoregressive Conditional Heteroskedasticity). In the presence of ARCH, the residuals themselves have an autoregressive representation:

$$\varepsilon_t = \alpha_0 + \alpha_1 \varepsilon_{t-1} + \dots + \alpha_n \varepsilon_{t-n} \quad (5.4.4)$$

ARCH can thus be tested by estimation of equation (5.4.4) with a prespecified lag order n . Subsequently the coefficients $\alpha_i \forall i=1, \dots, n$ are tested on significance using a LM test. The test statistic is χ^2 -distributed with n degrees of freedom. If the test statistic is above the critical value, the coefficients α_i in equation (5.4.4) are significant and an MA-part should be added to the model. Test statistics in table 5.23 are calculated by `JMulti` for lags from 1 to 5. Alternatively the test of White (1980) can be used

to examine the residuals on heteroskedasticity. We find heteroskedasticity only in the residuals of SAL.

| Series | Mean | σ | Skewness | Kurtosis |
|--------|--------|----------|----------|----------|
| PCI | 0.0000 | 0.0903 | 0.4794 | 1.15 |
| EXC | 0.0000 | 0.0274 | -0.0511 | 1.3464 |
| IND | 0.0000 | 0.0379 | -0.1335 | 0.0380 |
| SAL | 0.0000 | 0.0909 | -0.0578 | -0.0518 |

Table 5.20: Univariate models: Skewness and kurtosis

| Series | Test statistic | CV (99%) |
|--------|----------------|----------|
| PCI | 11.9703 | 9.2103 |
| EXC | 9.981 | 9.2103 |
| IND | 0.3996 | 9.2103 |
| SAL | 0.0674 | 9.2103 |

Table 5.21: Univariate models: Jarque-Bera test

| Series | Lag | | | | |
|-----------------|--------|--------|---------|---------|---------|
| | 1 | 2 | 3 | 4 | 5 |
| PCI | 0.0128 | 0.0132 | 0.0419 | 0.0804 | 0.3013 |
| EXC | 0.0409 | 0.5757 | 2.2220 | 2.4138 | 2.4189 |
| IND | 0.0000 | 0.0263 | 0.3475 | 0.3476 | 0.4387 |
| SAL | 0.0062 | 0.0076 | 0.0366 | 3.4300 | 3.4877 |
| CV (95%) | 3.8415 | 5.9915 | 7.8147 | 9.4877 | 11.071 |
| CV (99%) | 6.6349 | 9.2103 | 11.3449 | 13.2767 | 15.0863 |

Table 5.22: Univariate models: Box-Pierce test

| Series | Lag | | | | |
|-----------------|---------|---------|---------|---------|---------|
| | 1 | 2 | 3 | 4 | 5 |
| PCI | 2.4883 | 11.5169 | 12.0451 | 12.6325 | 13.4076 |
| EXC | 0.9267 | 8.4323 | 8.7798 | 8.7974 | 15.1851 |
| IND | 2.4413 | 3.5160 | 3.6484 | 3.5121 | 9.7773 |
| SAL | 10.6703 | 10.5682 | 16.6300 | 21.2555 | 21.2214 |
| CV (95%) | 3.8415 | 5.9915 | 7.8147 | 9.4877 | 11.071 |
| CV (99%) | 6.6349 | 9.2103 | 11.3449 | 13.2767 | 15.0863 |

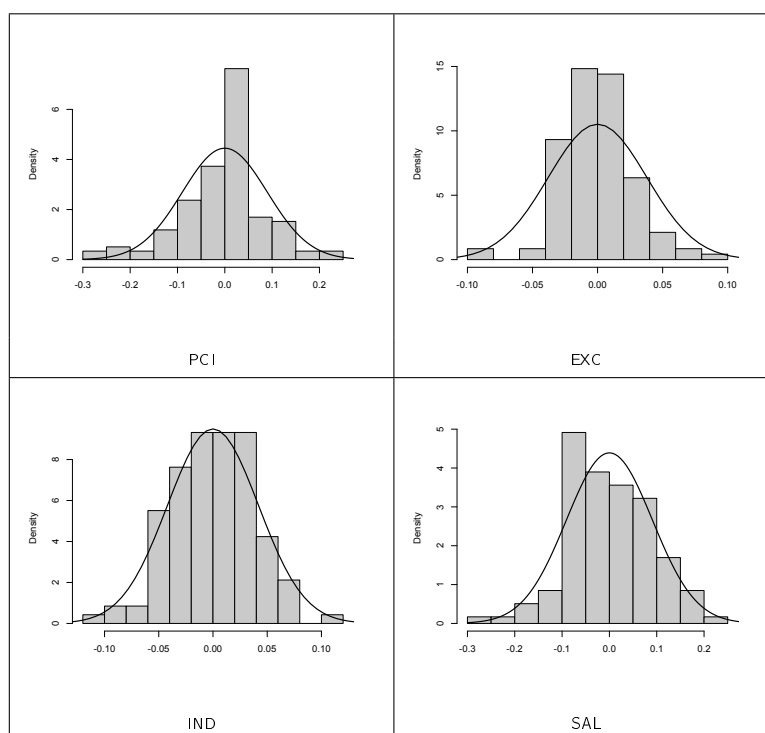
Table 5.23: Univariate models: ARCH-LM test (χ^2 - statistic)

Figure 5.10: Histogram residuals

Source: Authors' design

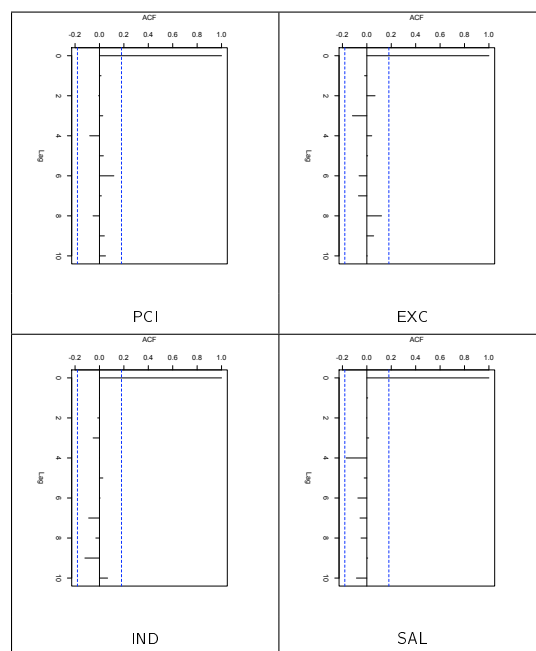


Figure 5.11: Univariate models: Autocorrelation residuals

Source: Authors' design

5.5 Cointegration and NN-VEC

In this section a NN-VEC including the three variables IND, SAL and EXC is constructed. Afore in section 5.5.1 the cointegration relationship is calculated. In the subsequent sections the NN-VEC including this cointegration relationship is estimated and used for prediction. Via cross-validation it is also examined, how many hidden neurons should be included in the model.

5.5.1 The Cointegration Relationship

The cointegration relationship is represented by a (3×1) vector B , which is transposed and multiplied with the data vector. The output from this product is the stationary univariate series z_t . In general the existence of such a cointegration vector can be explained by the fact that the effects of the non-stationary time series cancel each other out. As OLS estimation did not result in any cointegration relationship, we use the 2SLS method instead. Therefore a structural equation system is constructed using the four variables IND, SAL, EXC and PCI:

$$IND_t = b_{11} \cdot SAL_t + b_{12} \cdot EXC_t + z_t \quad (5.5.1)$$

$$IND_t = b_{21} \cdot SAL_t + b_{22} \cdot PCI_t + u_t \quad (5.5.2)$$

The two equations can be interpreted as macroeconomic-environment (5.5.1) and capital-market-incentive driven (5.5.2) supply-demand equation concerning the German automobile industry production and the US market. IND and SAL are endogenous variables, PCI and SAL are exogenous variables. Only equation (5.5.1) is used as cointegration relationship in the following. This means, that the variable PCI is not directly involved in the cointegration relationship. However it is needed, because only the simultaneous treatment of both equations ((5.5.1) and (5.5.2)) leads to a stationary relationship in equation (5.5.1).

Both equations of the system are identified. In the first step the variable SAL is regressed on EXC and PCI. Subsequently in the second step IND is regressed on the estimated

\widehat{SAL} and EXC. For details about the 2SLS method see for example Moosmüller (2004) p.186. The following parameters result:

$$\begin{aligned}b_{11} &= 1.1515 \\b_{12} &= -0.1867\end{aligned}$$

Those results correspond with the economic considerations from section 5.1. The correlation of IND with SAL is positive, which means that an increase in SAL let also IND increase. In contrast the correlation of IND and EXC is negative, which describes the negative impact of the USD per EUR exchange rate discussed in section 5.1. In addition the coefficient of EXC is smaller than that of SAL. This is realistic as the impact of SAL on IND is more direct and therefore stronger.

The equilibrium relationship or attractor z_t is received by multiplication of the cointegration vector C with the vector of variables:

$$z_t = (IND_t, SAL_t, EXC_t) \cdot B = (IND_t, SAL_t, EXC_t) \begin{pmatrix} 1 \\ -1.1515 \\ 0.1867 \end{pmatrix} \quad (5.5.3)$$

Figure 5.12 shows a plot of z_t . The ADF as well as the RADF test for several lags indicate that z_t is stationary.

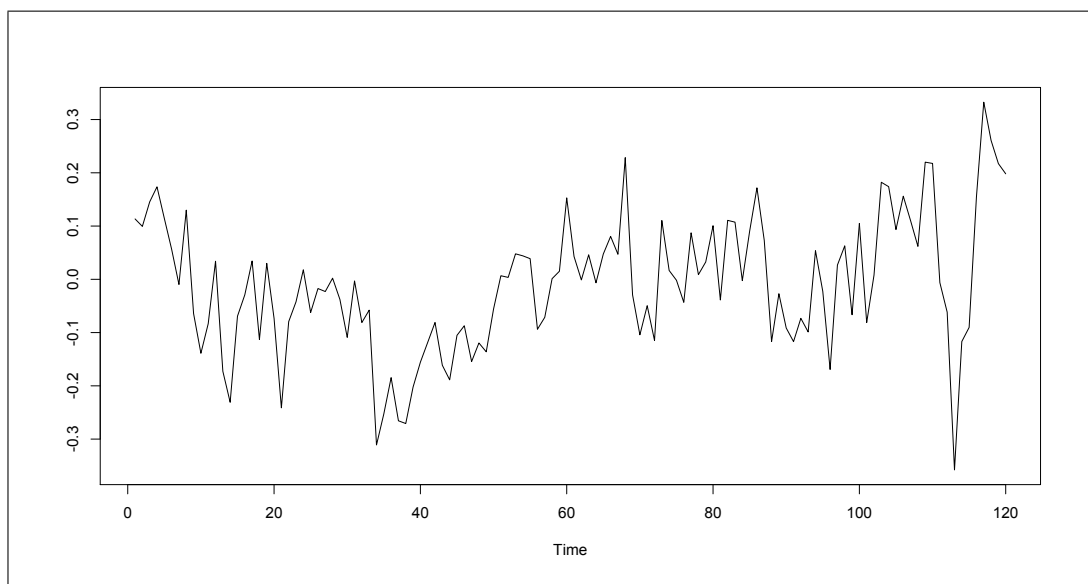


Figure 5.12: Cointegration relationship

Source: Author's design

5.5.2 Estimation of the NN-VEC

Before the NN-VEC can be estimated, the number of lags has to be determined for the common model. We use lag order $n=3$ for the common model. The cointegration relationship estimated in the previous section is used as z_t . The NN-VEC is written in vector representation (like equation (4.2.17)):

$$\begin{pmatrix} \Delta \text{IND}_t \\ \Delta \text{EXC}_t \\ \Delta \text{SAL}_t \end{pmatrix} = \sum_{i=1}^3 \begin{pmatrix} \alpha_{i11}^* & \alpha_{i12}^* & \alpha_{i13}^* \\ \alpha_{i21}^* & \alpha_{i22}^* & \alpha_{i23}^* \\ \alpha_{i31}^* & \alpha_{i32}^* & \alpha_{i33}^* \end{pmatrix} \begin{pmatrix} \Delta \text{IND}_{t-i} \\ \Delta \text{SAL}_{t-i} \\ \Delta \text{EXC}_{t-i} \end{pmatrix} + \begin{pmatrix} \alpha_{01} \\ \alpha_{02} \\ \alpha_{03} \end{pmatrix} + \begin{pmatrix} \alpha_{11} \\ \alpha_{21} \\ \alpha_{31} \end{pmatrix} z_{t-1} + \sum_{j=1}^h \tanh \left(\begin{pmatrix} \gamma_{01j} \\ \gamma_{02j} \\ \gamma_{03j} \end{pmatrix} \right) + \begin{pmatrix} \gamma_{11j} \\ \gamma_{12j} \\ \gamma_{13j} \end{pmatrix} z_{t-1} \beta_j + \begin{pmatrix} \varepsilon_{1t} \\ \varepsilon_{2t} \\ \varepsilon_{3t} \end{pmatrix}$$

This equation can be split into 3 independent equations with scalar outputs. This is necessary for estimation. Initial values are uniformly setted equal to 0.05, $\lambda = 0.0001$, $\tau = 100$, $\beta_1 = 6$ and $\beta_2 = 3$. The maximum number of iterations is 11. Again the first 120 values are used for estimation, the last 8 values for prediction. Keeping the maximum number of iterations small helps to avoid the problem of overlearning here (which in for univariate series is avoided by stopped training). Results for NN-VEC's with $h=0,1,2$ are shown in table 5.24 (results are calculated using the functions `estimate.NNVEC()` in appendix B.10 and `predict.NNVEC` in appendix B.11). The table contains also a linear VAR (estimated using the function `VAR()` in the package `vars`). Again one- and eight step prediction is considered using the same performance measures as in section 5.4.2.

The results for the individual series concerning out-of-sample prediction vary with the models in table 5.24. If we focus on the IND series, a NN-VEC(3) with $h = 1$ seems to be the best model. With increasing h the prediction performance concerning the IND series decreases. Results for the EXC series connected with that model are outperformed by a model with linear cointegration, which again is not as good as the VAR (one- and eight- step prediction). This let us assume that the EXC series is at least

weak exogenous, which means that it influences the system but is not influenced by the system. The EXC variable can be excluded by setting the parameters connecting the cointegration relationship with EXC equal to zero. However doing so would also influence the prediction for the other models as the multivariate model is highly interconnected. Thus we proceed with the analysis of the NN-VEC(3) with $h = 1$. In figure 5.13 its out-of-sample performance as well as the out-of-sample performance of the VAR is plotted. Parameters are shown in table 5.25.

If we compare the NN-VEC's to the univariate AR-NN's (especially IND in 5.12), then we observe, that for short predictions (one step) the univariate model perform better, whereas in the eight step prediction the NN-VEC delivers better predictions. The reason might be that univariate AR-NN are especially able to deal with short-run effects, whereas the NN-VEC can treat long-run effects more effective (this was already explained in section 5.1). Thus it depends of the prediction horizon which kind of model should be used.

| | In-sample | Out-of-sample 1 step prediction | | Out-of-sample 8 step prediction | |
|------------------------|------------------|--|------------------|--|------------------|
| Series | RMSE | RMSE | Theil IEC | RMSE | Theil IEC |
| Linear VAR | | | | | |
| IND | 0.0396 | 0.1167 | 1.2745 | 0.0838 | 1.2132 |
| SAL | 0.0857 | 0.0611 | 2.1523 | 0.2363 | 1.0215 |
| EXC | 0.0279 | 0.0016 | 0.0387 | 0.0327 | 1.1022 |
| NN-VEC with h=0 | | | | | |
| IND | 0.0369 | 0.0707 | 0.7721 | 0.0667 | 0.9657 |
| SAL | 0.0813 | 0.0054 | 0.1902 | 0.2332 | 1.0081 |
| EXC | 0.0277 | 0.0298 | 0.721 | 0.0343 | 1.1562 |
| NN-VEC with h=1 | | | | | |
| IND | 0.0411 | 0.0667 | 0.7284 | 0.0517 | 0.7485 |
| SAL | 0.0811 | 0.006 | 0.2114 | 0.2375 | 1.0267 |
| EXC | 0.0277 | 0.1938 | 4.6887 | 0.1836 | 6.1888 |
| NN-VEC with h=2 | | | | | |
| IND | 0.0367 | 0.1194 | 1.304 | 0.0625 | 0.9049 |
| SAL | 0.0813 | 0.0131 | 0.4615 | 0.2343 | 1.0128 |
| EXC | 0.0272 | 0.0292 | 0.7065 | 0.0247 | 0.8326 |

Table 5.24: Cointegrated NN with varying h

$$\begin{pmatrix} \alpha_{111}^* & \alpha_{112}^* & \alpha_{113}^* \\ \alpha_{121}^* & \alpha_{122}^* & \alpha_{123}^* \\ \alpha_{131}^* & \alpha_{132}^* & \alpha_{133}^* \end{pmatrix} = \begin{pmatrix} -0.3143 & -0.0921 & 0.0526 \\ 0.0180 & 0.5500 & 0.0943 \\ -0.0848 & 0.1295 & 0.0465 \end{pmatrix}$$

$$\begin{pmatrix} \alpha_{211}^* & \alpha_{212}^* & \alpha_{213}^* \\ \alpha_{221}^* & \alpha_{222}^* & \alpha_{223}^* \\ \alpha_{231}^* & \alpha_{232}^* & \alpha_{233}^* \end{pmatrix} = \begin{pmatrix} -0.0334 & -0.1220 & 0.0161 \\ -0.0605 & -0.0546 & 0.0072 \\ 0.0522 & 0.1144 & -0.0424 \end{pmatrix}$$

$$\begin{pmatrix} \alpha_{311}^* & \alpha_{312}^* & \alpha_{313}^* \\ \alpha_{321}^* & \alpha_{322}^* & \alpha_{323}^* \\ \alpha_{331}^* & \alpha_{332}^* & \alpha_{333}^* \end{pmatrix} = \begin{pmatrix} 0.2236 & 0.5686 & 0.3274 \\ 0.1253 & -0.0267 & -0.2104 \\ -0.1702 & -0.0480 & 0.0357 \end{pmatrix}$$

$$(\alpha_{01}, \alpha_{02}, \alpha_{03})^\top = (-0.9170, -5.9974, -5.9966)^\top$$

$$(\alpha_{11}, \alpha_{12}, \alpha_{13})^\top = (0.1551, 12.2202, 2.9086)^\top$$

$$(\gamma_{01}, \gamma_{02}, \gamma_{03})^\top = (-4.1134, 0.2569, 0.0098)^\top$$

$$(\gamma_{11}, \gamma_{12}, \gamma_{13})^\top = (0.6906, -29.7890, -4.3797)^\top$$

Table 5.25: Parameters NN-VEC(3)

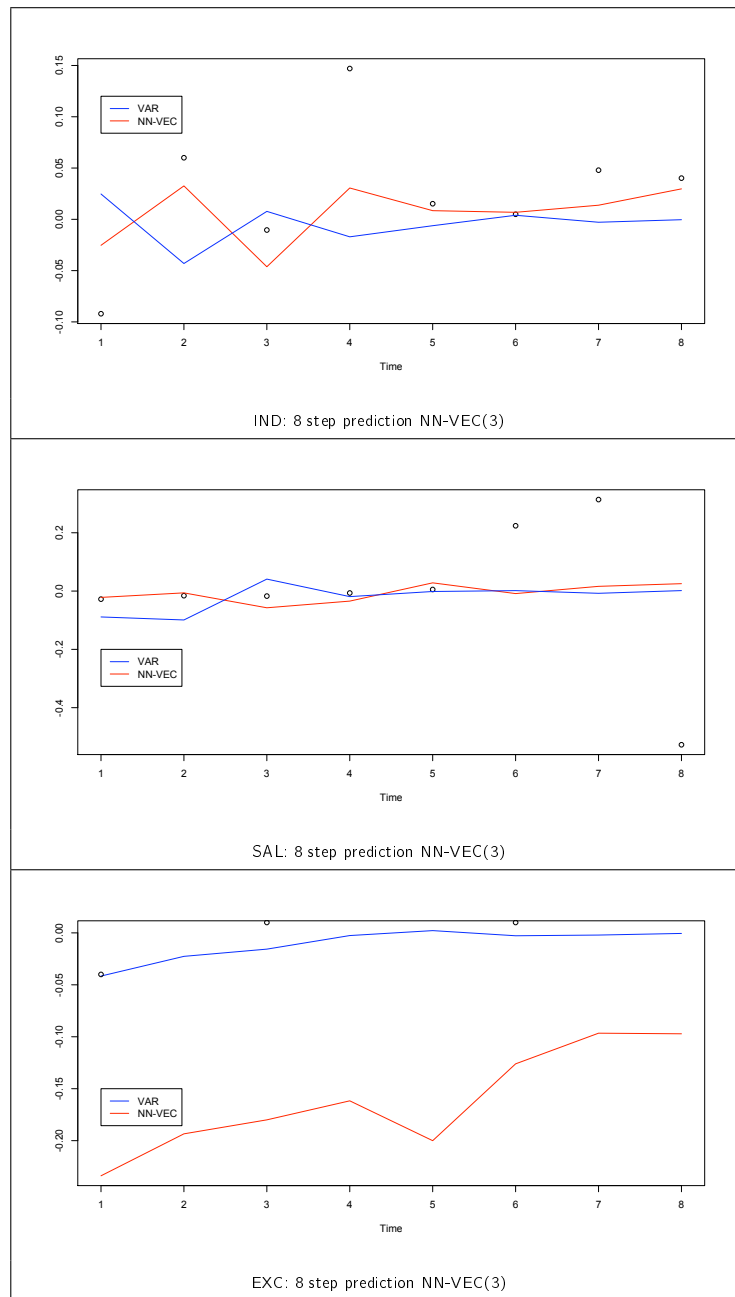


Figure 5.13: NN-VEC out-of-sample plots
Source: Authors' design

5.5.3 Residual Analysis

Again like in section 5.4.4 the assumptions on the residuals are checked. Therefore the same methods are used. Residuals of the IND and SAL series seem to fit at the i.i.d. Gaussian WN assumption, whereas the residuals of the EXC series does not. Consequently the model could be estimated again with different settings for the algorithm. Alternatively the EXC series could be excluded from the cointegration relationship.

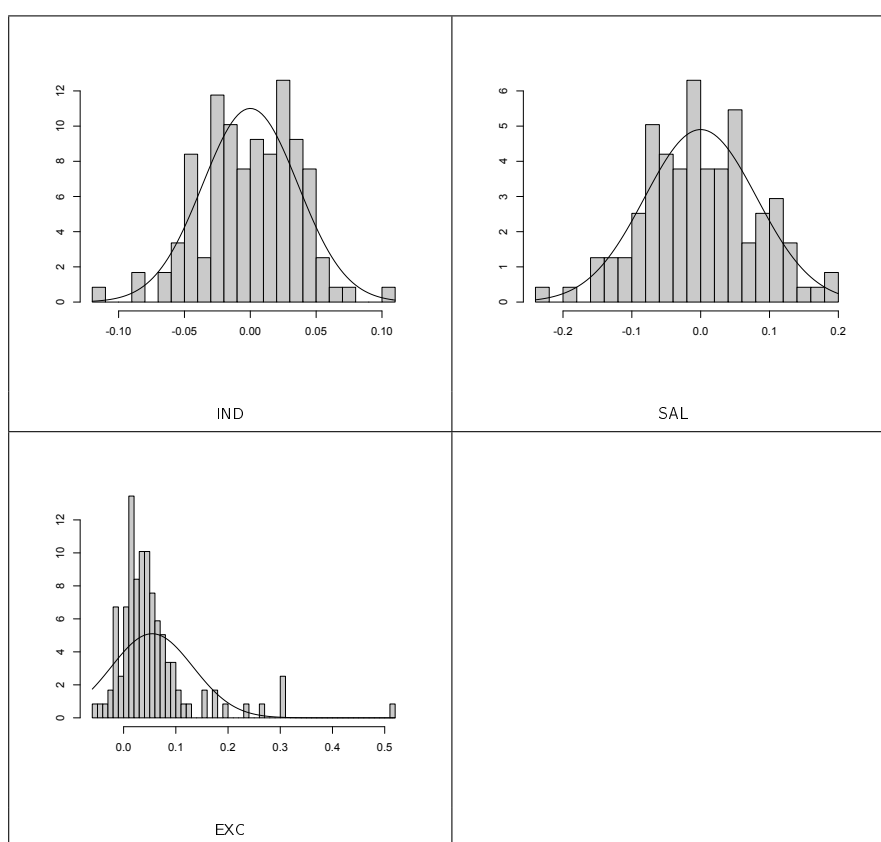


Figure 5.14: Histogram residuals NN-VEC(3)

Source: Authors' design

| Series | Mean | σ | Skewness | Kurtosis |
|--------|--------|----------|----------|----------|
| IND | 0.0000 | 0.0362 | -0.2181 | 0.2999 |
| SAL | 0.0000 | 0.0814 | -0.0144 | -0.2025 |
| EXC | 0.0547 | 0.0782 | 2.7748 | 10.6639 |

Table 5.26: NN-VEC(3): Skewness and kurtosis

| Series | Test statistic | CV (99%) |
|--------|----------------|----------|
| IND | 1.5965 | 9.2103 |
| SAL | 0.1232 | 9.2103 |
| EXC | 745.3444 | 9.2103 |

Table 5.27: NN-VEC(3): Jarque-Bera test

| Series | Lag | | | | |
|-----------------|---------|--------|---------|---------|---------|
| | 1 | 2 | 3 | 4 | 5 |
| IND | 0.2728 | 0.3738 | 0.3810 | 0.3992 | 0.4155 |
| SAL | 0.0013 | 0.003 | 0.0592 | 0.5259 | 0.8982 |
| EXC | 25.0724 | 29.619 | 30.5732 | 30.6633 | 30.6752 |
| CV (95%) | 3.8415 | 5.9915 | 7.8147 | 9.4877 | 11.071 |
| CV (99%) | 6.6349 | 9.2103 | 11.3449 | 13.2767 | 15.0863 |

Table 5.28: NN-VEC(3): Box-Pierce test

| Series | Lag | | | | |
|-----------------|--------|---------|---------|---------|---------|
| | 1 | 2 | 3 | 4 | 5 |
| IND | 0.1648 | 0.5944 | 3.5675 | 4.0638 | 4.1823 |
| SAL | 2.1954 | 4.1663 | 7.5125 | 7.7133 | 8.1279 |
| EXC | 9.7267 | 13.4696 | 13.5108 | 13.4225 | 13.3803 |
| CV (95%) | 3.8415 | 5.9915 | 7.8147 | 9.4877 | 11.071 |
| CV (99%) | 6.6349 | 9.2103 | 11.3449 | 13.2767 | 15.0863 |

Table 5.29: NN-VEC(3): ARCH-LM test (χ^2 - statistic)

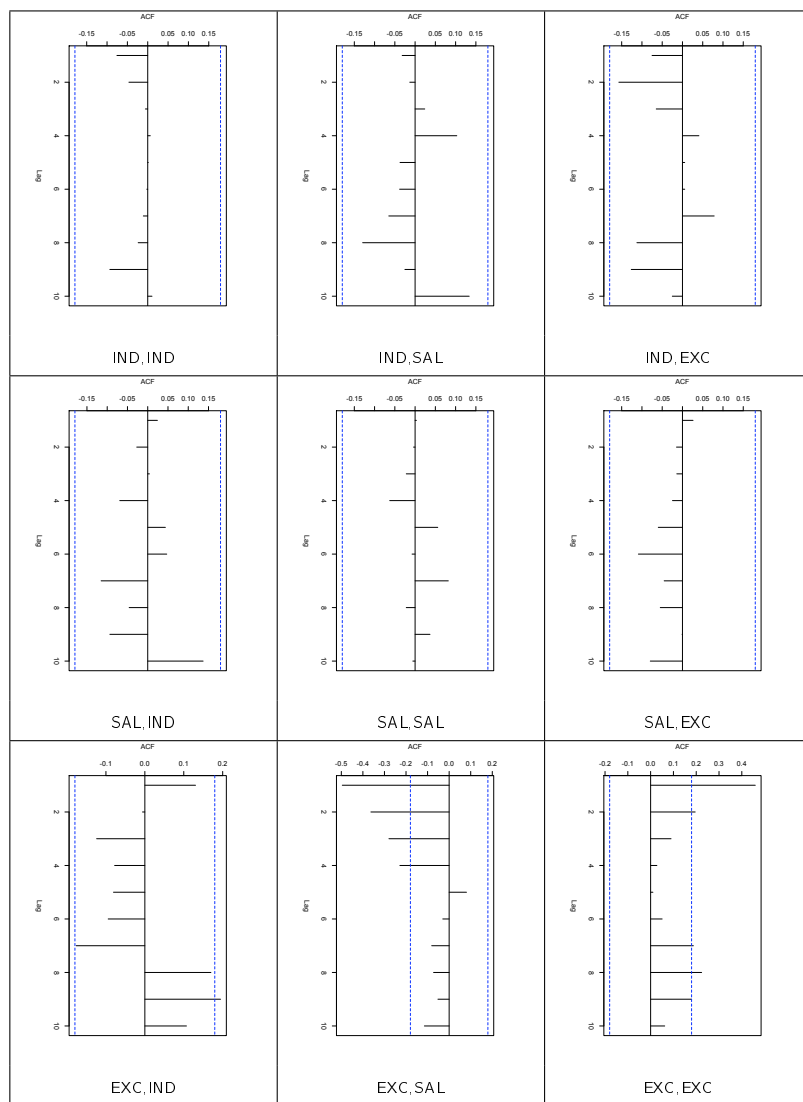


Figure 5.15: Auto- and cross-correlations NN-VEC(3)

Source: Authors' design

6 Conclusion

In this dissertation a method for analysis and prediction of nonlinear time series, AR-NN, based on neural networks is discussed. Nonlinear time series methods have to be applied, if any nonlinear effects or nonstandard features are in the data, which would disturb linear methods. Such effects occur especially with economic crises as it is shown in the empirical part. AR-NN combine the properties parametric and universal approximation, which makes them applicable at any nonlinear effect, as well as easily handable concerning model validation and parameter tests. Alternative models, in contrast, are either nonparametric (kernel regression) or dedicated to special nonlinearities like structural breaks (threshold regression). In the empirical part it is shown, that AR-NN outperform many linear and nonlinear methods concerning out-of-sample prediction, particularly in one-step prediction.

Further AR-NN are extended to multivariate error correction models based on the error correction theorem of Escribano and Mira (2002). This model combines a linear long-run equilibrium with nonlinear adjustment. If some linearly cointegrated series are nonlinear, such nonlinear error correction is essential. The linear cointegration relationship is estimated using a structural model and the 2SLS estimation method. This means that time series analysis here is combined with structural equation models. This is not yet very common in literature but might be an interesting point of research. The NN-VEC as proposed here, combined with structural equations, are especially useful for prediction of time series involved in a supply-demand system. The example in the empirical part is such a system, whereas the demand for foreign cars in the USA, the industry production of the car manufacturing industry in Germany (corresponds to supply) and the USD/EUR exchange rate are examined.

Various methods from existing literature for data preparation, variable selection, model estimation and models validation are brought together here with neural network theory to provide an adequate toolkit for our AR-NN models. Therefore some of those methods are modified, especially the MIC in combination with the method of Hausser and Strimmer (2009) or the early stopping method, which is extended for a search of

a global minimum within a finite number of iterations. In this field no theory exists in literature about multivariate neural networks, thus VAR-NN and NN-VEC are newly introduced by this dissertation.

Future topics of research based on this dissertation might be a comprehensive study about nonlinearities and economic crises, which would include data from various fields of economics including crises (financial data as stock indices as well as macroeconomic data as growth of GDP). The aim of such a study should be to show if crises cause nonlinearities in the data. The statistical tool therefore can be the nonlinearity test of Teräsvirta, Lin and Granger (1993). Furthermore the NN-VEC can be extended to a model which includes a nonlinear cointegration relationship. Although such a model might be more complex than the one in this dissertation, it might be possible to include a broader range of cointegration relationships and provide better predictions for variables which are only nonlinear cointegrated. Finally, the comparison between the models from this dissertation and linear as well as nonlinear alternatives could be continued, using other data sets from various fields of application, especially to see if there is any supply-demand relationship.

A Proof of Theorem 2.1

For this proof see Hornik (1993) p.1071. At first let us give a short nonformal description of the proof: We define a subset of $\mathcal{G}(\Psi; \mathcal{B}, \mathcal{W})$ such that \mathcal{W} contains a neighborhood of the origin and \mathcal{B} is compact and nondegenerate (see the requirements concerning those intervals in theorem 2.1) and make use of an extension of the dual-space argument of Cybenko (1989), which says that, if the selected subset should not be dense in $\mathcal{F}(\mathbf{X})$, a certain nonzero measure may exist. We show that no such measure exists for any selected subset of $\mathcal{G}(\Psi; \mathcal{B}, \mathcal{W})$ and thus prove the theorem.

Let X_{t-1} be a nondegenerate compact interval, $X_{t-1} \subset \mathbb{R}^n$. $|\cdot|$ denotes the determinant. The ϵ -neighborhood of S is denoted by $\mathbf{X}^\epsilon = \{S : |S - X_{t-1}| \leq \epsilon \text{ for } X_{t-1} \in \mathbf{X}\}$. Let $W(X_{t-1})$ be a function on the interval $[-1; 1]$,

$$W(X_{t-1}) = \begin{cases} c \cdot e^{-\frac{1}{1-|X_{t-1}|^2}} & |X_{t-1}| < 1 \\ 0 & |X_{t-1}| \geq 1. \end{cases} \quad (\text{A.0.1})$$

where c is to choose such that $\int_{\mathbb{R}^n} W(X_{t-1}) dX_{t-1} = 1$. With $\epsilon_0 > 0$ we can formulate the ϵ -mollifier of Ψ ,

$$J_\epsilon \Psi(X_{t-1}) = \int_{|U| \leq 1} W(U) \Psi(X_{t-1} - \epsilon U) dU. \quad (\text{A.0.2})$$

Concerning the mollifier the following fact holds, that if Ψ is Riemann integrable on \mathbf{X}^ϵ , then it is possible to approximate $J_\epsilon \Psi$ uniformly on \mathbf{X} by linear combinations of functions $\Psi(X_{t-1} - S)$ for $|S| \leq \epsilon$. (See lemma 4 in Hornik (1993) p.1070).

Let the vector $M = \max\{|X_{t-1}| : X_{t-1} \in \mathbf{X}\}$. Choose the initial sets \mathcal{W}_0 , \mathcal{B}_0 and some numbers ϵ_0 and η_0 such that $\mathcal{W}_0 = \{\Gamma : |\Gamma| < \eta_0\} \subseteq \mathcal{W}$ and $\mathcal{B}_0^{\eta_0 M + \epsilon_0} \subset \mathcal{B}$. Further Ψ should be nonpolynomial on \mathcal{W}_0 . According to the dual space argument, if

the subset $\mathcal{G}(\Psi; \mathcal{B}_0^{\epsilon_0}, \mathcal{W}_0)$ of $\mathcal{G}(\Psi; \mathcal{B}, \mathcal{W})$ is not dense in $\mathcal{F}(\mathbf{X})$, a nonzero signed finite measure ζ has to exist such that

$$\int_{\mathbf{X}} J_{\epsilon} \Psi(\gamma_0 + \Gamma^{\top} X_{t-1}) d\zeta(X_{t-1}) = 0 \quad (\text{A.0.3})$$

$\forall \Gamma \in \mathcal{W}_0, \gamma_0 \in \mathcal{B}_0$ and $0 < \epsilon < \epsilon_0$. We take the partial derivatives of order α for (A.0.3) and set $\Gamma = 0$:

$$\nabla^{\alpha} J_{\epsilon} \Psi(\gamma_0) \int_{\mathbf{X}} X_{t-1}^{\alpha} d\zeta(X_{t-1}) = 0 \quad (\text{A.0.4})$$

Lemma 5 in Hornik (1991) p.1071 states that for all $\alpha \geq 0$ always a $\gamma_0 \in \mathcal{B}_0$ and $0 < \epsilon < \epsilon_0$ can be found such that $\nabla^{\alpha} J_{\epsilon} \Psi(\gamma_0) \neq 0$. This means that $\int_{\mathbf{X}} X_{t-1}^{\alpha} d\zeta(X_{t-1})$ has to be zero if equation (A.0.4) is true. This is only the case if $\zeta = 0$ and thus theorem 2.1 is proved.

B R-Code

For calculation of the results the statistical programming language R is used. In the following sections a short description inclusive the code of the used functions is provided.

B.1 Lag Partition Matrix

We need a function which generates a matrix of the lagged values of the time series x_t for the calculations in the following. This matrix X has dimension $(n \times T)$, with the i th lagged series in the i th row, $i = 1, \dots, n$:

$$X = \begin{pmatrix} x_{1-1} & x_{2-1} & \dots & x_{T-1} \\ x_{1-2} & x_{2-2} & \dots & x_{T-2} \\ \vdots & \vdots & & \vdots \\ x_{1-n} & x_{2-n} & \dots & x_{T-n} \end{pmatrix} \quad (\text{B.1.1})$$

A problem is that we have only data for x_t , $t = 1, \dots, T$ but not for $t < 1$ as we need it for the matrix X . Hence we propose to substitute those missing values by the mean of the first five values of x_t . The following function generates X :

```

1 lag<-function(x,n)
2 ##x:=Time series vector
3 ##n:=Number of lags
4 {
5   X<-matrix(mean(x[1:5]),n,length(x))
6   for(i in 1:n) X[i,(i+1):(length(x))]<-x[1:(length(x)-i)]
7   return(X)
8 }
```

B.2 Polynomial Approximation Based Lag Selection

Like in the previous section polynomial approximation is used. The function in the following returns the AIC and BIC for selected lags up to lag order 4 using equation (3.1.3).

```
1 Polyapprox<-function(x,n)
2   ##x:=Time series vector
3   ##n:=Number of lags
4   {
5     if(n>4) return("Only up to 4 lags!")
6     m<-ifelse(n==1,1,(fak(n+3-1)/(fak(3)*fak(n-1))))
7
8     ##Preparation of the lags
9     X<-lag(x,4)
10    X[4,]<-switch(n,0,0,0,X[4,])
11    X[3,]<-switch(n,0,0,X[3,],X[3,])
12    X[2,]<-switch(n,0,X[2,],X[2,],X[2,])
13
14    ##The function
15    f1<-(x~X[1,]*X[2,]*X[3,]*X[4,])
16    r1<-lm(f1)
17
18    ##Return
19    return(list("AIC"=AIC(r1),"BIC"=AIC(r1,k=log(n+1+m))))
20  }
```

Example

Let `ind` be a vector containing the 128 observations of the industry production data (IND). The result for polynomial approximation based lag selection for lag order 4 is calculated by

```
Polyapprox(ind,4)
```

B.3 The MIC

This function returns the MIC using the method of Hausser and Strimmer (2009) as discussed in section 3.3.

```
1 ##Required packages
2 library(entropy)
3
4 MIC<-function(x,n,d)
5 ##x:=Time series vector
6 ##n:=Number of lags
7 ##d:=Number of bins
8 {
9   X<-lag(x,n)
10
11   return("MIC"=sqrt(1-exp(-2* mi.shrink
12                     (rbind(hist(x,breaks=d)$density,
13                             hist(X[n,],breaks=d)$density))))))
14 }
```

Example

The MIC for the ind data with lag 4 and 100 bins is calculated by

```
MIC(ind,4,100)
```

B.4 The Levenberg-Marquardt Algorithm for Univariate Models

This function executes the Levenberg-Marquardt algorithm for a fixed number of iterations including stopped-training (section 3.3.7). It returns the optimal parameter vector as well as many more information which are necessary for related functions in the following.

```
1 ##Required packages
2 library(numDeriv)
3 estimate.ARNN<-
4     function(x,n,h,iter,lambda,tau,init,partition)
5 ##x:=Time series vector
6 ##n:=Number of lags
7 ##h:=Number of hidden neurons
8 ##iter:=Number of iterations
9 ##lambda:=Paramter
10 ##tau:=Parameter
11 ##init:=Initial values for the nonlinear part
12 ##partition:=The relation to split the series in ES and VS
13 {
14 ##Initialization of input variables, ES and VS
15 u<-ceiling(length(x)*partition)
16 X<-lag(x,n)
17 b<-X[1:n,1:u]
18 b2<-x[1:u]
19 y<-x[(u+1):length(x)]
20
21 ##Number of parameters
22 r=(n+1)*h+(n+1)+h
23
24 ##Initialization parameter vector
25 a<-rep(1,r)
26 fit<-arima(b2,c(n,0,0))
27 la<-c(coef(fit)[n+1],coef(fit)[1:n])
```

```
28 a<-c(la,rep(init,(r-n-1)))
29 neu<-a
30
31 ##Initialization internal storage
32 h1<-matrix(NA,(iter+1),length(a))
33 h2<-matrix(NA,(iter+1),length(y))
34 v1<-rep(NA,(iter+1))
35 v2<-rep(NA,(iter+1))
36
37 ##Constructs for internal usage
38 pv<-rep(NA,length(y))
39 c<-rep(NA,n)
40 e2<-0
41 s1<-0
42 if(h>0)
43 {
44   q=0
45   s1<-rep(NA,h)
46   s1[1]<-(n+2)
47   if(h>1) for(i in 2:h) s1[i]<-(s1[i-1]+(n+2))
48
49 ##Basic components of functions
50   e2<-expression(for(i in s1)
51                   {
52                     +(tanh(crossprod(b,a[i:(i+n-1)])
53                       +a[i+n]))*a[i+n+1]
54                   })
55   }
56   e1<-expression(a[1]+crossprod(b,a[2:(n+1)]))
57
58 ##General functions
59 f<-function(s1,a) eval(e1)+eval(e2)
60 ff<-function(a) eval(e1)
61
```

```
62 ##Function with respect to the network weights
63 f1<-function(a) eval(e1)+eval(e2)
64
65 ##Difference between real and estimated values
66 f2<-function(a) b2-f1(a)
67
68 ##Performance function
69 f3<-function(a) sum((f2(a))^2)
70
71 ##Function for prediction
72 f4<-function(a,b) eval(e1)+eval(e2)
73
74 ##Execution of the algorithm
75 for(i in 1:(iter+1))
76 {
77   m1<-c(b2[(length(b2)-n+1):length(b2)],pv)
78   for(j in 1:length(y))
79     {
80       m1[j+n]<-f4(a,m1[(j+n-1):(j)])[1]
81     }
82   v1[i]<-(f3(a))
83   v2[i]<-sum((y-m1[(n+1):(length(m1))])^2)
84   h1[i,1:(length(a))]<-a
85   h2[i,1:(length(y))]<-m1[(n+1):(length(m1))]
86   i=i+1
87   if(f3(neu)==f3(a))
88     {
89       t=1
90     }
91   else
92     {
93       if(f3(neu)<(f3(a)))
94         {
95           t=tau
```

```

96     }
97     else
98     {
99         t=(1/tau)
100    }
101    }
102    a<-neu
103    neu<-a-crossprod(t(solve((crossprod(jacobian(f2,a))+
104    (lambda*t)*diag(r)))),crossprod((jacobian(f2,a)),f2(a)))
105  }
106
107  ##Iteration where an optimum is reached
108  for(i in 5:(iter))
109  {
110    if(v2[i]==min(v2[5:(iter)])) mini=i
111  }
112
113  ##Function return
114  return(list("Minimum VS-RSS reached at"=mini,
115            "Minimum VS-RSS"=v2[mini],
116            "Minimum ES-RSS"=v1[mini],
117            "Optimal parameter vector"=h1[mini,],
118            "f"=f,"ff"=ff,"f4"=f4,"Data"=x,"Lags"=n,
119            "Support variable"=s1,"ES"=u))
120  }

```

Example

Let the total estimation subset TS be the first 120 values of the `ind` data. A object ARNN1 for the `ind` data with $n=4$, $h=2$, $\lambda = 1$, $\tau = 100$, the maximal number of iteration $i^{max} = 100$, $ES/TS=0.95$ and initial values for the nonlinear part uniformly one is calculated by

```
ARNN1<-estimate.ARNN(ind[1:120],4,2,100,1,100,1,0.95)
```


B.5 Residuals ES

This function returns the residuals of an estimated AR-NN (only ES residuals).

```
1 residuals.ARNN<-function(l)
2 ##l:=Result generated by function estimate.ARNN
3 {
4 return(l$"Data"[1:l$"ES"]-l$"f"(l$"Support variable",
5         l$"Optimal parameter vector"))
6 }
```

Example

ES-residuals for the object ARNN1 are calculated by

```
residuals.ARNN(ARNN1)
```

The in-sample RMSE is calculated by

```
sqrt(sum((residuals.ARNN(ARNN1))^2)/(120*0.95))
```

B.6 Fitted Values ES

This function returns the fitted values of an estimated AR-NN (only ES fitted values).

```
1 fitted.ARNN<-function(l)
2   ##l:=Result generated by function estimate.ARNN
3   {
4   return(l$"f"(l$"Support variable",l$"Optimal parameter vector"))
5   }
```

Example

ES-fitted values for the object ARNN1 are calculated by

```
fitted.ARNN(ARNN1)
```

B.7 Prediction

This function returns one- and more-step predictions for an estimated AR-NN.

```
1 predict.ARNN<-function(l,k)
2 ##l:=Result generated by function estimate.ARNN
3 ##k:=Steps to predict
4 {
5   c<-rep(NA,n)
6   a<-l$"Optimal parameter vector"
7   n<-l$"Lags"
8   m<-c(l$"Data"[(length(l$"Data")-n+1):
9         length(l$"Data")],rep(NA,k))
10
11  for(j in 1:k) m[j+n]<-l$"f4"(a,m[(j+n-1):j])
12
13  return(m[(n+1):(length(m)]))
14 }
```

Example

An one-step prediction out of the object ARNN1 is calculated by

```
predict.ARNN(ARNN1,1)
```

B.8 The Covariance Matrix

This function returns the NIC, the Wald test statistics for each parameter or the covariance matrix of an estimated AR-NN.

```
1 covariance.ARNN
2     <-function(l,h,type=c("NIC","Wald","Covariance"))
3 ##l:=Result generated by function estimate.ARNN
4 ##h:=Number of hidden neurons
5 {
6 ##Input elements
7   x<-l$"Data"[1:l$"ES"]
8   n<-l$"Lags"
9   r=(n+1)*h+(n+1)+h
10  a<-l$"Optimal parameter vector"[1:r]
11
12 ##Network function
13   if(h>0)
14     {
15       s1<-l$"Support variable"[1:h]
16       f<-expression(l1$"f"(s1,a))
17       f1<-function(a) eval(f)
18     }
19   else
20     {
21       f1<-l$"ff"
22     }
23
24 ##RSS function
25   f2<-function(a) 0.5*sum((x-f1(a)[1:l$"ES"])^2)
26
27 ##Submatrices
28   Gamma<-function(a)
29     (1/length(x))*hessian(f2,a)
30   Upsilon<-function(a)
```

```
31         (1/length(x))*(grad(f2,a)%*%t(grad(f2,a)))
32
33 ##Covariance matrix
34 C<-Upsilon(a)%*%solve(Gamma(a))%*%Upsilon(a)
35
36 Wald<-a^2/diag(C)
37
38 NIC<-(1/length(x))*(f2(a)+(1/length(x))
39         *sum(diag(Upsilon(a)%*%solve(Gamma(a))))))
40
41 ##Return
42 if(type=="Wald")
43   {
44     return(list("Wald statistic"=Wald))
45   }
46 if(type=="NIC")
47   {
48     return(list("NIC"=NIC))
49   }
50 if(type=="Covariance")
51   {
52     return(list("Covariance matrix"=C))
53   }
54 }
```

Example

The NIC for the object ARNN1 is calculated by

```
covariance.ARNN(ARNN1,2,type="NIC")
```

B.9 The Lee-White-Granger Test

This function executes the Lee-White-Granger test for additional hidden nonlinearity (see section 3.4.1.1 and Lee, White and Granger (1993)).

```

1 LWG.test<-function(l)
2  ##l:=Result generated by function estimate.ARNN
3  {
4  ##Preparation of the lagged matrix
5    n<-l$"Lags"
6    x<-l$"Data"[1:(length(l$"Data")*l$"ES/TS")]
7    X<-lag(x,n)
8    a<-l$"Optimal parameter vector"
9    f1<-l$"f1"
10
11  ##Number of paramters
12    m<-ifelse(n==1,1,(fak(n+3)/(fak(3)*fak(n))-n-1))
13
14  ##Preparation of the input for the polynomial term
15    Y<-lag(x,4)
16    Y[4,]<-switch(n,0,0,0,Y[4,])
17    Y[3,]<-switch(n,0,0,Y[3,],Y[3,])
18    Y[2,]<-switch(n,0,Y[2,],Y[2,],Y[2,])
19
20  ##Residuals of the restricted function
21    r1<-x-f1(a)
22
23  ##Residuals of the unrestricted function
24    f2<-(r1~jacobian(f1,a)+Y[1,]*Y[2,]*Y[3,]+Y[4,])
25    r2<-lm(f2)
26
27  ##The test statistics
28    T1<-length(x)*(sum(fitted(r2))^2
29      /sum((r1)^2))
30    T2<-((sum((r1)^2)-sum((residuals(r2))^2))/m)/

```

```
31         (sum((residuals(r2))^2)/(length(x)-n-m))
32
33 ##Return
34     return(list("Chi-square statistic"=
35     c("Test statistic"=T1,"df"=m, "Critical value"
36     =qchisq(0.95,m)), "F statistic"=c("Test statistic"
37     =T2, "df1"=m, "df2"=(length(x)-n-m), "Critical
38     value"=qf(0.95,df1=m,df2=(length(x)-n-m))))
39 }
```

Example

The Lee-White-Granger test statistic for the object ARNN1 is calculated by

```
LWG.test(ARNN1)
```

B.10 Estimation of the NN-VEC

This function estimates for each variable separately the parts of a three dimensional NN-VEC.

```
1 estimate.NNVEC
2     <-function(nr,X,cv,n,h,iter,lambda,tau,init,ab)
3 ##nr:=Number of the equation (variable) to display
4 ##X:=Data matrix with 3 variables in rows
5 ##cv:=Cointegration vector
6 ##h:=Number of hidden neurons
7 ##n:=Number of lags
8 ##iter:=Number of iterations
9 ##lambda:=Parameter
10 ##tau:=Parameter
11 ##init:=Initial parameter vector
12 ##ab:=(1xh) vector with beta weights
13 {
14 ##Initialization of input variables
15   x1<-X[1,]
16   x2<-X[2,]
17   x3<-X[3,]
18
19   z1<-diff(x1)
20   z2<-diff(x2)
21   z3<-diff(x3)
22
23   y1<-x1[1:(length(x1)-1)]
24   y2<-x2[1:(length(x2)-1)]
25   y3<-x3[1:(length(x3)-1)]
26
27   Z1<-lag(z1,n)
28   Z2<-lag(z2,n)
29   Z3<-lag(z3,n)
30 }
```



```
31 b<-rbind(Z1,Z2,Z3)
32 bb<-rbind(y1,y2,y3)
33
34 b2<-diff(X[nr,])
35
36 ##Number of parameters
37 r=2*h+2+n*3
38
39 ##Initialization parameter vector
40 a<-t(matrix(init,1,r))
41 neu<-a
42
43 ##Initialization internal constructs
44 s<-c(1,3,5,7)
45 ak<-c(ab[1],0,ab[2],0,ab[3],0,ab[4])
46
47 ##Basics for network functions
48 e1<-expression(
49   t(b)%*%a[1:(n*3),]+
50   a[n*3+1,]+t(a[n*3+2,]%*%(cv%*%bb))+
51   for(i in 1:s[h])
52     {
53       +tanh(a[n*3+2+i,]+
54         t(a[n*3+3+i,]%*%(cv%*%bb)))*ak[i]
55     })
56
57 e2<-expression(
58   t(b)%*%a[1:(n*3),]+
59   a[n*3+1,]+t(a[n*3+2,]%*%(cv%*%bb)))
60
61 e<-ifelse(h>0,e1,e2)
62
63 ##Definition network function respective the weights
64 f1<-function(a) eval(e)
```

```

65
66 ##Difference between real and estimated values
67   f2<-function(a) b2-f1(a)
68
69 ##Performance function
70   f3<-function(a) sum((f2(a))^2)
71
72 ##Execution of the algorithm
73   for(i in 1:(iter+1))
74     {
75       i=i+1
76       t<-ifelse(f3(neu)>=f3(a),tau,(1/tau))
77       a<-neu
78       neu<-a-crossprod(t(solve((crossprod
79         (jacobian(f2,a))+(lambda*t)*diag(r)))),
80         crossprod((jacobian(f2,a)),f2(a)))
81     }
82
83 ##Return
84 return(list("Minimal RSS"=f3(neu),
85 "Optimal parameter vector"=neu,
86 "f"=function(a,b,bb,g) eval(e),"Cointegration relationship"
87 =cv,"Beta vector"=ak,"DiffData"=b,"LevelData"=bb,
88 "Lags"=n,"Support variable"=s,"Residuals"=f2(neu)))
89 }

```

Example

Again the TS contains 120 values. Let the cointegration vector be $(1, -1.1515, 0.1867)^\top$ and the β -values $\beta_1 = 6$, $\beta_2 = 3$, $\beta_3 = 1.5$, $\beta_4 = 0.1$. Objects NNVEC1, NNVEC2 and NNVEC3 for the three series using separate equations (like equation (4.2.15) and (4.2.16)) with $n=2$, $h=4$, $\lambda = 0.0001$, $\tau = 100$, the maximal number of iteration $i^{max} = 11$, and initial values uniformly 0.05 are calculated by

```
X<-rbind(ind[1:120],sal[1:120],exc[1:120])
```

```
cv<-c(1,-1.1515,0.1867)
ab<-c(6,3,1.5,0.1)
NNVEC1<-estimate.NNVEC(1,X,cv,2,4,100,0.0001,100,0.05,ab)
NNVEC2<-estimate.NNVEC(2,X,cv,2,4,100,0.0001,100,0.05,ab)
NNVEC3<-estimate.NNVEC(3,X,cv,2,4,100,0.0001,100,0.05,ab)
```

B.11 Prediction with the NN-VEC

This function can be used with three separate models from the previous section to calculate predictions out of a joint multivariate model.

```
1 predict.NNVEC<-function(l1,l2,l3,k)
2   ##l1,l2,l3:=Results generated by estimate.NNVEC
3   ##k:=Steps to predict
4   {
5     ##Initialization of input variables
6     n<-l1$"Lags"
7     b<-l1$"DiffData"
8     bb<-l1$"LevelData"
9     ak<-l1$"Beta vector"
10    f<-l1$"f"
11
12    a<-t(rbind(t(l1$"Optimal parameter vector"),
13              t(l2$"Optimal parameter vector"),
14              t(l3$"Optimal parameter vector")))
15    g<-t(matrix(0,3,1))
16
17    ##Initialization internal constructs
18    t=0
19    pv<-rep(NA,k,3)
20    c<-rep(NA,length(b[,1]))
21    cc<-rep(NA,3)
22    l=0
23    s<-l1$"Support variable"
24
25    ##Prediction algorithm
26    m1<-matrix(NA,3,k)
27    m2<-b[,length(b[1,])]
28    m3<-bb[,length(bb[1,])]
29    for(j in 1:k)
30      {
```

```
31         m1[,j]<-f(a,m2,m3,g)
32         m2[4:length(m2)]<-m2[1:(length(m2)-3)]
33         m2[1:3]<-m1[,j]
34         m3<-m3+m1[,j]
35     }
36
37 ##Return predicted values
38     return(m1,a)
39 }
```

Example

For forecasting a joint model is used. The basis are the three objects from the previous section. For example a two step forecast is calculated by:

```
predict.NNVEC(NNVEC1,NNVEC2,NNVEC3,2)
```

Bibliography

- Akaike, H. (1974): *A New Look at the Statistical Model Identification* in: IEEE Transactions on Automatic Control, vol. 19, pp. 716–723.
- Al-Ballaa, N.R. (2005): *Test for Cointegration Based on Two-Stage Least Squares* in: Journal of Applied Statistics, vol. 32, pp. 707 – 713.
- Amemiya, T. (1980): *Selection of Regressors* in: International Economic Review, vol. 21, pp. 331–354.
- Anders, U. (1997): *Statistische Neuronale Netze* Doctoral Dissertation, University of Karlsruhe.
- Anders, U., Korn, O. and Schmitt, C. (1998): *Improving the Pricing of Options: A Neural Network Approach* in: Journal of Forecasting, vol. 17, pp. 369–388.
- Auestad, B. and Tjøstheim, D. (1990): *Identification of Nonlinear Time Series: First Order Characterization and Order Determination* in: Biometrika, vol. 77, pp. 669–687.
- Baum, E.B. and Haussler, D. (1988): *What Size Net Gives Valid Generalization?* in: Neural Computation, vol. 1, pp. 151–160.
- Bishop, C.M. (1995): *Neural Networks for Pattern Recognition* Clarendon Press, Oxford.
- Bottou, L. (2003): *Stochastic Learning* in: Bousquet, O., Luxburg, U.v. and Rätsch, G., editors: *Advanced Lectures on Machine Learning* Springer, Berlin et al., pp. 146–168.
- Box, G.E.P. and Jenkins, G.M. (1976): *Time Series Analysis - Forecasting and Control* 2nd edition. Holden-Day, San Francisco et al.
- Burnham, K.P. and Anderson, D.R. (2004): *Multimodel Inference - Understanding AIC and BIC in Model Selection* in: Sociological Methods & Research, vol. 33, pp. 261–304.
-

-
- Carathéodory, C. (1927): *Vorlesungen über Reelle Funktionen* 2nd edition. AMS Chelsea Publishing, American Mathematical Society; Reprint (Oktober 2004).
- Castro, J.L., Mantas, C.J. and Benítez (2000): *Neural Networks with a Continuous Squashing Function in the Output are Universal Approximators* in: *Neural Networks*, vol. 13, pp. 561–563.
- Chakraborty, K. et al. (1992): *Forecasting the Behaviour of Multivariate Time Series Using Neural Networks* in: *Neural Networks*, vol. 5, pp. 961–970.
- Chan, K.S. and Tong, H. (1985): *On the Use of the Deterministic Lyapunov Function for the Ergodicity of Stochastic Difference Equations* in: *Advances in Applied Probability*, vol. 17, pp. 666–678.
- Cybenko, G. (1989): *Approximation by Superposition of a Sigmoidal Function* in: *Mathematics of Control, Signals, and Systems*, vol. 2, pp. 303–314.
- Davidson, R. and MacKinnon, J.G. (1993): *Estimation and Inference in Econometrics* Oxford University Press, New York et al.
- Dickey, D.A. and Fuller, W.A. (1979): *Distribution of the Estimators for Autoregressive Time Series With a Unit Root* in: *Journal of the American Statistical Association*, vol. 74, pp. 427–431.
- Dufrenot, G. and Mignon, V. (2002): *Recent Developments In Nonlinear Cointegration With Applications To Macroeconomics And Finance* Kluwer Academic Publishers, Dordrecht.
- Dutta, S., Ganguli, R. and Samanta, B. (2005): *Investigation of two Neural Network Methods in an Automatic Mapping Exercise* in: *Applied GIS (Online Journal)* vol. 1.
- El Ayech, H. and Trabelsi, A. (2007): *Decomposition Method for Neural Multiclass Classification Problem* in: *International Journal of Applied Mathematics and Computer Sciences*, vol. 3, pp. 207–210.
- Engle, R. (1982): *Autoregressive Conditional Heteroskedasticity with Estimates of the Variance of United Kingdom Inflation* in: *Econometrica*, vol. 50, pp. 987–1008.
-

-
- Engle, R.F. and Granger, C.W. (1987): *Co-Integration and Error Correction: Representation, Estimation, and Testing* in: *Econometrica*, vol. 55, pp. 251–276.
- Escribano, A. and Mira, S. (2002): *Nonlinear Error Correction Models* in: *Journal of Time Series Analysis*, vol. 23, pp. 509–522.
- Evans, M.K. (2003): *Practical Business Forecasting* Blackwell Publishers Ltd., Oxford et al.
- Fan, J. and Yao, Q. (2003): *Nonlinear Time Series* Springer, New York et al.
- Fletcher, R. and Reeves, C.M. (1964): *Function Minimization by Conjugate Gradients* in: *The Computer Journal*, vol. 7, pp. 149–154.
- Fonseca, G. and Tweedie, R.L. (2002): *Stationary Measures for Non-Irreducible Non-Continuous Markov Chains with Time Series Applications* in: *Statistica Sinica*, vol. 12, pp. 651–660.
- Fuller, W.A. (1976): *Introduction to Statistical Time Series* Wiley, New York et al.
- Funahashi, K. (1989): *On the Approximate Realization of Continuous Mappings by Neural Networks* in: *Neural Networks*, vol. 2, pp. 183–192.
- Gencay, R. (1994): *Nonlinear Prediction of Noisy Time Series with Feedforward Networks* in: *Physics Letters A*, vol. 187, pp. 397–403.
- Granger, C.W. (1988): *Some Recent Developments in A Concept of Causality* in: *Journal of Econometrics*, vol. 39, pp. 199–211.
- Granger, C.W. and Hallman, J.J. (1988): *The Algebra of I(1) Series* in: *Finance and Economics Discussion Series, Board of Governors of the Federal Reserve System* vol. 45.
- Granger, C.W. and Lin, J.L. (1994): *Using the Mutual Information Coefficient to Identify Lags in Nonlinear Models* in: *Journal of Time Series Analysis*, vol. 15, pp. 371–383.
- Granger, C.W. and Newbold, P. (1974): *Spurious Regression in Econometrics* in: *Journal of Econometrics*, vol. 2, pp. 110–120.
-

- Granger, C.W. and Teräsvirta, T. (1993): *Modelling Nonlinear Economic Relationships* Oxford University Press, Oxford.
- Hagan, M.T. and Menhaj, M.B. (1994): *Training Feedforward Networks with the Marquardt Algorithm* in: IEEE Transactions on Neural Networks, vol. 5, pp. 989–993.
- Haigh, J. (2010): *Introduction to Markov Chains - The Finite Case* in: Significance, vol. 7, pp. 88–89.
- Hallman, J.J. (1990): *Nonlinear Integrated Series, Cointegration and Application* PhD Dissertation, University of California San Diego.
- Hamilton, J.D. (1994): *Time Series Analysis* Princeton University Press, Princeton.
- Handelsblatt (2009a): *Mini Wird ein Wenig Deutsch* <http://www.handelsblatt.com/unternehmen/industrie/mini-wird-ein-wenig-deutsch;2459431>, downloaded 2nd December 2009.
- Handelsblatt (2009b): *Daimler und sein Sparzwang* <http://www.handelsblatt.com/meinung/kommentar-unternehmen/verlagerung-c-klasse-daimler-und-sein-sparzwang;2493244>, downloaded 2nd December 2009.
- Harbour, R. and Joas, A. (2008): *How the Weak Dollar Is Reviving U.S. Manufacturing* in: Oliver Wyman Journal, no volume, pp. 66–71.
- Härdle, W., Kleinow, T. and Tschernig, R. (2001): *Web Quantlets for Time Series Analysis* in: Annals of the Institute of Statistical Mathematics, vol. 53, pp. 179–188.
- Hassler, U. (2007): *Stochastische Integration und Zeitreihenmodellierung* Springer, Berlin et al.
- Hatanaka, M. (1996): *Time-Series-Based Econometrics - Unit Roots and Co-Integrations* Oxford University Press, Oxford.
- Hausser, J. and Strimmer, K. (2009): *Entropy Inference and the James-Stein Estimator, with Application to Nonlinear Gene Association Networks* in: Journal of Machine Learning Research, vol. 10, pp. 1469–1484.
-

-
- Haykin, S. (2009): *Neural Networks and Learning Machines* 3rd edition. Pearson Education, Upper Saddle River et. al.
- Hestenes, M.R. and Stiefel, E. (1952): *Methods of Conjugate Gradients for Solving Linear Systems* in: *Journal of Research of the National Bureau of Statistics*, vol. 49, pp. 409–436.
- Hornik, K. (1991): *Approximation Capabilities of Multilayer Feedforward Networks* in: *Neural Networks*, vol. 4, pp. 251–257.
- Hornik, K. (1993): *Some New Results on Neural Network Approximation* in: *Neural Networks*, vol. 6, pp. 1069–1072.
- Hornik, K., Stinchcombe, M. and White, H. (1989): *Multilayer Feedforward Networks Are Universal Approximators* in: *Neural Networks*, vol. 2, pp. 359–366.
- Huang, W. et al. (2006): *Selection of the Appropriate Lag Structure of Foreign Exchange Rates Forecasting Based on Autocorrelation Coefficient* in: *Lecture Notes in Computer Science*, vol. 3973, pp. 512–517.
- Hush, D.R. and Salas, J.M. (1988): *Improving the Learning Rate of Back-Propagation with the Gradient Reuse Algorithm* in: *IEEE International Conference on Neural Networks*, vol. 1, pp. 441–447.
- Hutchinson, J.M. (1994): *A Radial Basis Function Approach to Financial Time Series* PhD Dissertation, Massachusetts Institute of Technology.
- Inoue, A. and Kilian, L. (2006): *On the Selection of Forecasting Models* in: *Journal of Econometrics*, vol. 130, pp. 273–306.
- Jacobs, R.A. (1988): *Increased Rates of Convergence Through Learning Rate Adaption* in: *Neural Networks*, vol. 1, pp. 295–307.
- Johansen, S. (1995): *Likelihood-Based Inference in Cointegrated Vector Autoregressive Models* Oxford University Press, Oxford et al.
- Judge, G.G. et al. (1984): *The Theory and Practice of Econometrics* Wiley, New York et al.
- Kaasra, I. and Boyd, M. (1996): *Designing a Neural Network for Forecasting Financial and Economic Time Series* in: *Neurocomputing*, vol. 10, pp. 215–236.
-

- Kuan, C.-M. and White, H. (1994): *Artificial Neural Networks: An Econometric Perspective* in: *Econometric Reviews*, vol. 13, pp. 1 – 91.
- Lee, T.H., White, H. and Granger, C.W. (1993): *Testing for Neglected Nonlinearity in Time Series Models* in: *Journal of Econometrics*, vol. 56, pp. 269–290.
- Leisch, F., Trapletti, A. and Hornik, K. (1998): *On the Stationarity of Autoregressive Neural Network Models* in: Report Series Wirtschaftsuniversität Wien Report No.21.
- Levenberg, K. (1944): *A Method for the Solution of Certain Non-Linear Problems in Least Squares* in: *Quarterly Journal of Applied Mathematics*, vol. 2, pp. 164–168.
- Liao, Y., Fang, S.C. and Nuttle, H.L.W. (2003): *Relaxed Conditions for Radial-Basis Function Networks to be Universal Approximators* in: *Neural Networks*, vol. 16, pp. 1019–1028.
- Lin, F. et al. (1995): *Time Series Forecasting with Neural Networks* in: *Complexity International - Online Journal*, vol. 2, <http://www.complexity.org.au/ci/vol02/>.
- Lütkepohl, H. and Tschernig, R. (1996): *Nichtparametrische Verfahren zur Analyse und Prognose von Finanzmarktdaten* in: Bol, G., Nakhaeizadeh, G. and Vollmer, K.-H., editors: *Finanzmarktanalyse und -Prognose mit Innovativen Quantitativen Verfahren* Physica-Verlag, Heidelberg, pp. 145–171.
- Marquardt, D.W. (1963): *An Algorithm for Least-Squares Estimation of Nonlinear Parameters* in: *Journal of the Society of Industrial and Applied Mathematics*, vol. 11, pp. 431–441.
- Medeiros, M.C., Teräsvirta, T. and Rech, G. (2006): *Building Neural Network Models for Some Series: A Statistical Approach* in: *Journal of Forecasting*, vol. 25, pp. 49–75.
- Meyn, S.P. and Tweedie, R.L. (1993): *Markov Chains and Stochastic Stability* Springer, London et al..
- Mishkin, F.S. (1996): *Understanding Financial Crises: A Developing Country Perspective* NBER Working Paper Nr. 5600.
-

- Mohatarem, G.M. (2003): *Impact of the Strong Dollar on the US Auto Industry* in: Bergsten, F. and Williamson, J., editors: *Institute for International Economics Special Report No. 16: Dollar Overvaluation and the World Economy* Institute for International Economics, Washington, pp. 135–144.
- Moody's (2008): *Global Automotive Manufacturer Outlook June 2008* http://www.kisrating.com/report/moodys_report/?????/global%2020080630.pdf, downloaded 2nd December 2009.
- Moosmüller, G. (2004): *Empirische Wirtschaftsforschung* Pearson Studium, Munich et al.
- Murata, N., Yoshizawa, S. and Amari, S. (1994): *Network Information Criterion - Determining the Number of hidden Units for an Artificial Neural Network Model* in: *IEEE Transactions on Neural Networks*, vol. 5, pp. 865–872.
- Onoda, T. (1995): *Neural Network Information Criterion for the Optimal Number of Hidden Units* in: *Proceedings, IEEE International Conference on Neural Networks*, vol. 1, pp. 275–280.
- Oppner, M. (1998): *A Bayesian Approach to On-line Learning* in: Saad, D., editor: *On-line Learning in Neural Networks* Cambridge University Press, Cambridge et al, pp. 363–378.
- Polak, E. and Ribière, G. (1969): *Note sur la Convergence de méthodes de Directions Conjuguées* in: *Revue Française d'Informatique et de Recherche Opérationnelle*, vol. 16, pp. 35–43.
- Qi, M. and Zhang, G.P. (2001): *An Investigation of Model Selection Criteria for Neural Network Time Series Forecasting* in: *European Journal of Operational Research*, vol. 132, pp. 666–680.
- Raman, H. and Sunlikumar, N. (1995): *Multivariate Modelling of Water Resources Time Series Using Artificial Neural Networks* in: *Hydrological Sciences*, vol. 40, pp. 145–163.
- Rech, G., Teräsvirta, T. and Tschernig, R. (2001): *A Simple Variable Selection Technique for Nonlinear Models* in: *Communications in Statistics, Theory and Methods*, vol. 30, pp. 1227– 1241.
-

- Resnick, S.I. (1992): *Adventures in Stochastic Processes* Birkhäuser, Boston et al.
- Rumelhardt, D.E., Hinton, G.E. and Williams, R.J. (1986a): *Learning Representations by Back-Propagating Errors* in: *Nature*, vol. 323, pp. 533–536.
- Rumelhart, D.E., Hinton, G.E. and Williams, R.J. (1986b): *Chapter 8: Learning Internal Representation by Error Propagation* in: Rumelhart, D.E. and McClelland, J.L., editors: *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, vol. 1: Foundations* Cambridge, MA, pp. 310–362.
- Said, S.E. and Dickey, D.A. (1984): *Testing for Unit Roots in Autoregressive Moving Average Models of Unknown Order* in: *Biometrika*, 71, pp. 599–607.
- Schlittgen, R. and Streitberg, B.H.J. (1995): *Zeitreihenanalyse* 6th edition. Oldenbourg, Munich et al.
- Schraudolph, N.N. (2002): *Fast Curvature Matrix-Vector Products for Second-Order Gradient Descent* in: *Neural Computation*, vol. 4, pp. 1723–1738.
- Schwarz, G. (1978): *Estimating the Dimension of a Model* in: *The Annals of Statistics*, vol. 6, pp. 461–464.
- Shannon, C.E. (1948): *A Mathematical Theory of Communication* in: *The Bell System Technical Journal*, vol. 27, pp. 379–423.
- Silverman, B. (1986): *Density Estimation for Statistics and Data Analysis* Chapman and Hall, London et al.
- Steurer, E. (1996): *Prognose von 15 Zeitreihen der DGOR mit Neuronalen Netzen* in: *OR Spektrum*, vol. 18, pp. 117–125.
- Stone, M.H. (1948): *The Generalized Weierstrass Approximation Theorem* in: *Mathematics Magazine*, vol. 21, pp. 237–254.
- Swanson, N. R. and White, H. (1997): *A Model Selection Approach to Real-Time Macroeconomic Forecasting Using Linear Models and Artificial Neural Networks* in: *The Review of Economics and Statistics*, vol. 79, pp. 540–550.
- Teräsvirta, T., Lin, C.-F. and Granger, C.W. (1993): *Power of the Neural Network Linearity Test* in: *Journal of Time Series Analysis*, vol. 14, pp. 209–220.
-

- Humboldt Institution on Transatlantic Issues (2005): *The Twin Deficits in the United States and the Weak Dollar - Adjustments in the World Economy and Policy Recommendations for Germany and the European Union* Berlin.
- Tjøstheim, D. (1990): *Non-Linear Time Series and Markov Chains* in: *Advances in Applied Probability*, vol. 22, pp. 587–611.
- Tjøstheim, D. and Auestad, B. (1994): *Nonparametric Identification of Nonlinear Time Series- Selecting Significant Lags* in: *Journal of the American Statistical Association*, vol. 89, pp. 1410–1419.
- Trapletti, A., Leisch, F. and Hornik, K. (2000): *Stationary and Integrated Autoregressive Neural Network Processes* in: *Neural Computation*, vol. 12, pp. 2427–2450.
- Tschernig, R. (2005): *Nonparametric Time Series Analysis in JMulti* JMulti Online Help, <http://www.jmulti.com/download/help/nonpar.pdf>, downloaded 19th November 2009.
- Tschernig, R. and Yang, L. (2000): *Nonparametric Lag Selection for Time Series* in: *Journal of Time Series Analysis*, vol. 21, pp. 457– 487.
- VDA (2010): *Zahlen & Fakten - Jahreszahlen* <http://www.vda.de/de/zahlen/jahreszahlen/>, downloaded 3rd August 2010.
- Vogl, T.P. et al. (1988): *Accelerating the Convergence of the Back-Propagation Method* in: *Biological Cybernetics*, vol. 59, pp. 257–263.
- Wald, A. (1943): *Test of Statistical Hypotheses Concerning Several Parameters When the Number of Observations is Large* in: *Transactions of the American Mathematical Society*, vol. 54, pp. 426–482.
- Wei, W. (1990): *Time Series Analysis* Addison-Wesley, Redwood City et al.
- Weierstrass, K. (1885): *Über die Analytische Darstellbarkeit Sogenannter Willkürlicher Functionen einer Reellen Veränderlichen* in: *Sitzungsberichte der Königlich Preußischen Akademie der Wissenschaften zu Berlin*, vol. 2, pp. 633–639, 789–805.
- White, H. (1980): *A Heteroscedasticity-Consistent Covariance Matrix Estimator and a Direct Test for Heteroscedasticity* in: *Econometrica*, 48, pp. 817– 838.
-

-
- White, H. (1988): *Economic Prediction Using Neural Networks: The Case of IBM Daily Stock Returns* in: IEEE International Conference on Neural Networks 1988, vol. 2, pp. 451–458.
- White, H. (1989a): *An Additional Hidden Unit Test for Neglected Nonlinearity in Multilayer Feedforward Networks* in: Proceedings of the International Joint Conference on Neural Networks, Washington D.C., New York, vol. 2, pp. 90–131.
- White, H. (1989b): *Learning in Artificial Neural Networks: A Statistical Perspective* in: Neural Computation, vol. 1, pp. 425–464.
- White, H. (1989c): *Some Asymptotic Results for Learning in Single Hidden Layer Feedforward Network Models* in: Journal of the American Statistical Association, vol. 84, pp. 1003–1013.
- White, H. (1992): *Artificial Neural Networks: Approximation and Learning Theory* Blackwell, Oxford et al.
- White, H. and Domowitz, I. (1984): *Nonlinear Regression with Dependent Observations* in: Econometrica, vol. 52, pp. 143–161.
- Widmann, G. (2000): *Künstliche Neuronale Netze und ihre Beziehung zur Statistik* Doctoral Dissertation, University of Tübingen.
- Wold, H. (1954): *A Study in the Analysis of Stationary Time Series* Almqvist and Wiksell Book Co., Uppsala.
- Yule, U.G. (1927): *On a Method of Investigating Periodicities in Disturbed Series, with Special Reference to Wolfer's Sunspot Numbers* in: Philosophical Transactions of the Royal Society of London Series A, vol. 226, pp. 267–298.
-

Data for the empirical application have been downloaded at 4th december 2009. The exact links are:

- Data from Thomson One Banker are available at banker.thomsonib.com (Commercial website)
 - Exchange rate data: <http://www.bundesbank.de/statistik>
 - Industry production data: <http://www.bundesbank.de/statistik>
 - Car sales data: <http://www.bea.gov>
-

Index

- 2SLS, 103, 140
 - 3SLS, 103

 - AC, 48, 115, 136
 - Activation function, 19
 - ADF, 27, 33
 - AFPE, 55, 56, 115
 - AIC, 47, 53, 81, 86, 115
 - AR, 1, 6, 7, 10
 - ARCH, 136
 - Arithmetic mean, 21
 - ARMA, 1, 8
 - Augmented, 2, 11

 - Backpropagation, 66
 - Batch learning, 57
 - BIC, 47, 53, 86, 115
 - Bold-driver, 67
 - Box-Pierce test, 136
 - Bundesbank, 109

 - CAFPE, 56, 115
 - Chaos science, 9
 - Chaotic, 31
 - Characteristic polynomial, 32
 - Conditional expectation, 7
 - Conjugate gradient, 69, 71
 - Consistency, 82

 - Delta-bar-delta-rule, 67

 - Early stopping, 76
 - Econometrics, 1, 6
 - Environment measure, 17
 - ES, 76, 119, 136
 - EUR, 75, 105, 109

 - Feedforward, 3
 - FPE, 52, 54, 55
 - Frankfurt Stock Exchange, 109

 - Gauss-Newton, 73
 - Gaussian WN, 7, 8
 - Geometrical ergodic, 30
 - Geometrical ergodicity, 30
 - Gradient vector, 61, 66
 - Gradient-reuse, 68
 - Granger-causal, 89

 - Hessian matrix, 61, 63, 69–71, 85
 - Heteroscedasticity, 7, 136
 - Hidden neurons, 16

 - i.i.d., 7, 29
 - IC, 47
 - Integrable, 17
 - Interpretation of parameters, 121

 - Jacobian matrix, 61, 71, 72
 - James-Stein shrinkage estimator, 51

 - Lag selection, 48, 91
 - Landau symbol, 70
 - Laspeyres index, 109
 - Layer, 10
 - Levenberg-Marquardt, 71, 73, 119
 - Line search, 69
 - Line-search, 68, 69
 - LM test, 81, 136
 - LM tests, 41
 - LMD, 28
 - LMM, 28
 - Local-linear estimator, 54
-

-
- MA, 8
 - Maclaurin series, 38
 - Markov chain, 29, 30
 - MI, 49
 - MIC, 50, 115
 - MSE, 17, 120

 - Nadaraya-Watson estimator, 54
 - NED, 96–99
 - Newton direction, 70
 - NFPE, 115
 - NIC, 81, 86, 87
 - NLS, 59, 84, 86
 - Nondegenerate, 17
 - Nonlinear models, 9
 - Nonpolynomial, 16, 17
 - Nonstandard features, 9

 - OLS, 39, 119
 - On-line learning, 57
 - Oscillation, 67
 - Overfitting, 75
 - Overlearning, 75
 - Overparametrization, 18

 - PAC, 49, 115
 - Performance function, 58
 - Product rule, 72

 - Quasi-Newton, 70, 71

 - RADF, 27, 34, 105, 112
 - RBF, 19
 - Response-surface test, 104
 - Riemann integrable, 16, 19
 - RMSE, 80, 119
 - Roots, 31
 - RSS, 77

 - Scaling, 21
 - Shannon-entropy, 50

 - Shortcut connections, 15
 - Sigmoid, 20
 - Singularity, 72
 - SMD, 28, 30
 - SMM, 27, 30
 - STAR, 9, 23
 - State space, 30
 - Stationarity, 8, 27, 30
 - Steepest descent, 66, 68, 71
 - Stochastic learning, 57
 - Stochastic part, 8
 - Stopped training, 75, 76
 - Structural break, 23

 - tanh, 40
 - TAR, 9, 24
 - Taylor expansion, 38
 - Taylor polynomial, 38, 40
 - Theil IEC, 120
 - Threshold function, 23
 - Time series analysis, 6
 - Transition probability, 30

 - Universal approximation property, 16–18
 - UR, 31–33
 - USD, 75, 105, 109

 - VAR, 88, 91, 103
 - VEC, 95, 96, 98, 100, 103, 140
 - VS, 76, 119

 - Wald test, 82
 - Weakly stationary, 28, 33
 - Weight space, 17
 - WN, 88
 - Wold decomposition theorem, 8
-