A project completed as part of the requirements for the

BSc (Hons) Computer Studies

entitled

# "Neural Networks For Financial Time Series Prediction: Overview Over Recent Research"

by

Dimitri PISSARENKO

in the years

2001-2002

# Abstract

Neural networks are an artificial intelligence method for modelling complex target functions. During the last decade they have been widely applied to the domain of financial time series prediction and their importance in this field is growing. The present work aims at serving as an introduction to the domain of financial time series prediction, emphasizing the issues particularly important with respect to the neural network approach to this task. The work concludes with a discussion of current research topics related to neural networks in financial time series prediction.

# Contents

**Table of contents** 2

**List of tables** 6

**List of figures** 11

**Acknowledgements** 12

**1 Financial time series** 13

1.1 Properties of financial time series . . . . . . . . . . . . . . . . . . . 13

1.2 Efficient Market Hypothesis . . . . . . . . . . . . . . . . . . . . . . . 16

1.3 Data used for financial time series prediction . . . . . . . . . . . 18

1.3.1 Technical data . . . . . . . . . . . . . . . . . . . . . . . . . 19

1.3.2 Fundamental data . . . . . . . . . . . . . . . . . . . . . . . 20

1.3.3 Derived entities . . . . . . . . . . . . . . . . . . . . . . . . . 21

1.4 Classical methods for financial time series processing . . . . . . . 24

1.4.1 ARIMA process . . . . . . . . . . . . . . . . . . . . . . . . 25

1.4.2 ARCH models . . . . . . . . . . . . . . . . . . . . . . . . . 32

**2 Neural networks** 35

2.1 Introduction . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 35

2.1.1 What are neural networks? . . . . . . . . . . . . . . . . . . 35

2.1.2 Basic definitions . . . . . . . . . . . . . . . . . . . . . . . . 36

2.1.3 Properties of neural networks . . . . . . . . . . . . . . . . . 39

2.1.4 Multi-layer perceptron . . . . . . . . . . . . . . . . . . . . . 40

2.1.5 Learning algorithms . . . . . . . . . . 43

2.1.6 Stochastic gradient descent backpropagation learning algorithm . . . . . . . . . . 44

2.1.7 Scaled conjugate gradient learning algorithm . . . . . . . 51

2.2 Pre- and postprocessing . . . . . . . . . . 62

2.2.1 Curse of dimensionality . . . . . . . . . . 62

2.2.2 Operation of a neural network . . . . . . . . . . 64

2.2.3 Methods of pre- and postprocessing . . . . . . . . . . 65

2.2.4 Feature selection . . . . . . . . . . 66

2.2.5 Principal component analysis . . . . . . . . . . 70

2.2.6 Prior knowledge . . . . . . . . . . 74

2.3 Time series processing with neural networks . . . . . . . . . . 79

2.3.1 Overview . . . . . . . . . . 79

2.3.2 Multi-layer perceptron . . . . . . . . . . 80

2.3.3 Time-delay neural network . . . . . . . . . . 82

2.3.4 Jordan networks . . . . . . . . . . 83

2.3.5 Elman networks . . . . . . . . . . 83

2.3.6 Multi-recurrent networks . . . . . . . . . . 84

2.3.7 Other network architectures . . . . . . . . . . 87

**3 Financial time series prediction and ANNs** **89**

3.1 Historical Background . . . . . . . . . . 89

3.2 Application of neural networks in today's business . . . . . . . . 94

3.3 Examples of ANNs in finance . . . . . . . . . . 95

3.3.1 Example 1 . . . . . . . . . . 95

3.3.2 Example 2 . . . . . . . . . . 98

3.3.3 Example 3 . . . . . . . . . . 99

3.4 Design of ANNs in finance . . . . . . . . . . 102

3.4.1 Step 1: Variable selection . . . . . . . . . . 103

3.4.2 Step 2: Data collection . . . . . . . . . . 104

3.4.3 Step 3: Data preprocessing . . . . . . . . . . 105

3.4.4 Step 4: Data partitioning . . . . . . . . . . 107

3.4.5 Step 5: Neural network design . . . . . . . . . . . . . . . . 109
3.4.6 Step 6: Evaluation of the system . . . . . . . . . . . . . . . 114
3.4.7 Step 7: Training the ANN . . . . . . . . . . . . . . . . . . . 115
3.4.8 Step 8: Implementation . . . . . . . . . . . . . . . . . . . . 120

**4 Related Issues 122**

4.1 Impact of artificial intelligence to the practice of stock trading . 122
4.2 Integrated time series predictor . . . . . . . . . . . . . . . . . . . . 123
4.2.1 Introduction . . . . . . . . . . . . . . . . . . . . . . . . . . 123
4.2.2 Motivation . . . . . . . . . . . . . . . . . . . . . . . . . . . 124
4.2.3 Possible Implementations . . . . . . . . . . . . . . . . . . . 125
4.2.4 Justification of the integrated time series predictor . . . . 128
4.3 The perfection aspiration . . . . . . . . . . . . . . . . . . . . . . 131
4.3.1 Problem definition . . . . . . . . . . . . . . . . . . . . . . . 131
4.3.2 Counterargument 1: Artificial versus Natural Intelligence 132
4.3.3 Counterargument 2: The Engineering Approach to Development of Complex Systems . . . . . . . . . . . . . . . . 133
4.3.4 Counterargument 3: What for? . . . . . . . . . . . . . . . . 134
4.4 Problems of ANNs in finance . . . . . . . . . . . . . . . . . . . . . 135

**5 Conclusions 138**

**Bibliography 140**

**Glossary 155**

**Appendix 163**

**A AR example 164**

A.1 Notation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 164
A.2 Example . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 164

**B Regression analysis 167**

**C ARIMA example** **169**

**D Forecasting with ARIMA** **172**

**E Taylor series expansion** **175**

**F Confidence intervals** **176**

**G Kolmogorov's theorem** **178**

**H The $\mathcal{O}$ notation** **180**

**I Fisher's discriminant** **182**

I.1 Derivation for 2-classes case . . . . . . . . . . . . . . . . . . . . . . 182

I.2 Generalized version of Fisher's discriminant . . . . . . . . . . . . 186

# List of Tables

2.1 Reaction of the network approximating the OR function to different inputs . . . 43

C.1 AIC values for different ARIMA models . . . 171

D.1 Confidence intervals . . . 174

# List of Figures


1.1 Daily volatility of the Dow Jones index from December 3, 1962 to December 31, 1986 . . . . . 14

1.2 Daily volatility of the Dow Jones index from January 2, 1987 to June 6, 1988 . . . . . 15

1.3 Average volatility of stock prices around days on which firm specific news are announced (Donders and Vorst [1996]). . . . . . . . 16

1.4 Annual values of S&P 500 index (1871 to 1997) . . . . . 21

1.5 Annual S&P 500 real returns (1872 to 1995) . . . . . 22

1.6 Examples of AR processes with different parameters (in all cases there are 100 observations, $x_0 = 0.5$ and $\mu = 0.2$). Figure a) shows an $AR(1)$ process with $\gamma = 0.5$. Figure b) shows an $AR(2)$ process with $\gamma_1 = -0.25$, $\gamma_2 = -0.25$. Figure c) is an $AR(3)$ process with $\gamma_1$=-0.3, $\gamma_2 = -0.3$, $\gamma_2$=-0.3. . . . . . 27

1.7 Preparation of the time series for usage with the ARIMA model . 28

1.8 Visualisation of the SACFs of a stationary (a)) and a non-stationary process (b)) . . . . . 29

1.9 The forecasting process . . . . . 31

1.10 Monthly Excess Log US Stock Returns, 1926 to 1994 ([Campbell et al., 1997, p. 482]) . . . . . 33

2.1 Basic structure of a multi-layer perceptron . . . . . 37

2.2 Neural network with bias units. . . . . . 38

2.3 Data processing within an ANN node . . . . . 40

2.4 Binary, sigmoid and tangens hyperbolicus transfer functions . . . 41
2.5 ANN approximating the OR function . . . . . . . . . . . . . . . . 43
2.6 Basic principle of gradient descent (Schraudolph and Cummins [2002]). . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 45
2.7 Gradient descent: movement to the minimum (Schraudolph and Cummins [2002]). . . . . . . . . . . . . . . . . . . . . . . . . . . 46
2.8 Stochastic gradient descent backpropagation learning algorithm ([Mitchell, 1997, p. 98]) . . . . . . . . . . . . . . . . . . . . . . . . 47
2.9 Schematic illustration of fixed-step gradient descent for an error function which has substantially different curvatures along different directions ([Bishop, 1996, p. 265]). . . . . . . . . . . . . . . . . 51
2.10 Bracketing the minimum ([Bishop, 1996, p. 273]). . . . . . . . . . 53
2.11 Parabolic interpolation used to perform line-search minimization ([Bishop, 1996, p. 274]). . . . . . . . . . . . . . . . . . . . . . . . 54
2.12 Orthogonality of new gradient after line minimization ([Bishop, 1996, p. 275]). . . . . . . . . . . . . . . . . . . . . . . . . . . . 55
2.13 The concept of conjugate directions ([Bishop, 1996, p. 276]). . . 55
2.14 Scaled conjugate gradient learning algorithm (Møller [1993]) . . . 58
2.15 One way to specify a mapping from a $d$-dimensional space $x_1, \ldots, x_d$ to an output variable $y$ is to divide the input space into a number of cells, as indicated here for the case $d = 3$, and to specify the value of $y$ for each of the cells. The major problem with this approach is that the number of cells and hence the number of example data points required, grows exponentially with $d$, a phenomenon known as the curse of dimensionality ([Bishop, 1996, p. 8]). . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 63
2.16 Schematic illustration of the use of pre-processing and post-processing in conjunction with a neural network mapping ([Bishop, 1996, p. 296]). . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 64

2.17 Example of data from two classes as described by two feature variables $x_1$ and $x_2$. If the data was described by either feature alone then there would be strong overlap of the two classes, while if both features are used, as shown here, then the classes are well separated ([Bishop, 1996, p. 307]) . . . . . . . . . . . . . . . . . . 68
2.18 Sequential forward selection illustrated for a set of four features, denoted by 1,2,3 and 4. The single best feature variable is chosen first, and then features are added one at a time such that at each stage the variable chosen is the one which produces the greatest increase in the criterion function ([Bishop, 1996, p. 309]). . . . . 69
2.19 Examples of two-dimensional data sets which can be well approximated by a single parameter $\eta$ ([Bishop, 1996, pp. 314–315]) . . 72
2.20 Auto-associative neural network for dimensionality reduction ([Bishop, 1996, p. 315]). . . . . . . . . . . . . . . . . . . . . . . . . . . . 73
2.21 Fitting values and derivatives with tangent prop. Let $f$ be the target function for which three examples $x_1, x_2, x_3$ are known (a)). Based on these points the learner might generate the hypothesis $g$ (b)). If the derivatives are also known, the learner can state the more accurate hypothesis $h$ shown in figure c) ([Mitchell, 1997, p. 347]). . . . . . . . . . . . . . . . . . . . . . . . . . . . 76
2.22 Schematic architecture of a network for transition-invariant object recognition in two-dimensional images. In a practical system there may be more than two layers between the input and the outputs ([Bishop, 1996, p. 325]). . . . . . . . . . . . . . . . . . . 77
2.23 A feedforward neural network with time window as a non-linear AR model (Dorffner [1996], [Bishop, 1996, p. 303]) . . . . . . . . 81
2.24 Jordan network (Dorffner [1996]) . . . . . . . . . . . . . . . . . . 84
2.25 An example of an Elman network (Dorffner [1996]) . . . . . . . . 85
2.26 A multi-recurrent network (Ulbricht [1995], Dorffner [1996]) . . . 85

3.1 The five stages of neural network research development and its business impact (Smith and Gupta [2000]) . . . . . . . . . . . . . 91

3.2 Application domains of neural networks (Wong et al. [1997a]) . . 94
3.3 Sliding window technique used in the experiment of Tino et al. [2000] . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 96
3.4 Structure of the system described in Chenoweth and Obradovic [1996] . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 98
3.5 Design of the neural network in the agents presented in the study of Terna [2000] . . . . . . . . . . . . . . . . . . . . . . . . . . 100
3.6 Design process of a neural network based financial time series prediction system (Kaastra and Boyd [1996]). . . . . . . . . . . . 102
3.7 A walk-forward sliding windows testing routine (Kaastra and Boyd [1996]). . . . . . . . . . . . . . . . . . . . . . . . . . . . . 108
3.8 Distribution of the scaled values of the S&P 500 closing prices (1962-1993) time series. In figure a) the time series was scaled by means of minimum and maximum scaling, in figure b) the same time series was scaled with mean and standard deviation scaling (Kaastra and Boyd [1996]). . . . . . . . . . . . . . . . . . . . . . 114
3.9 Possible neural network training and testing set errors (Kaastra and Boyd [1996]). . . . . . . . . . . . . . . . . . . . . . . . . . 116
3.10 Simplified graphical representation of a neural network error surface (Kaastra and Boyd [1996]). . . . . . . . . . . . . . . . . . . 119

4.1 Logical structure of an integrated time series predictor . . . . . 126
4.2 Architecture of textual financial time series predictor described in Wüthrich et al. [1998]. . . . . . . . . . . . . . . . . . . . . . . 130

A.1 SACF of the residuals . . . . . . . . . . . . . . . . . . . . . . . 166

B.1 Observations of variables $x$ and $y$ . . . . . . . . . . . . . . . . . . 168
B.2 Fitted line so that the sum of squared residuals is minimized . . 168

C.1 SACFs of the original time series and the time series after the taking the differences . . . . . . . . . . . . . . . . . . . . . . . . 170

D.1 Taylor series expansion . . . . . . . . . . . . . . . . . . . . . . . 173

D.2 Script for the calculation of the $\varepsilon_t$ time series . . . . . . . . . . . 174
D.3 Script for the calculation of the confidence intervals . . . . . . . . 174

F.1 Probability density and the confidence intervals . . . . . . . . . . 177

I.1 Two projections of training samples onto a line. The samples indicated by empty and filled circles belong to two different classes. The projection b) yields a better class separation (discrimination) than a) (Gutierrez-Osuna [2001]). . . . . . . . . . . . . . . . . . . . 183
I.2 Projection on $x_2$ axis yields a better class separation in spite of the fact that separation on $x_1$ axis would result in a larger distance between means, since there is a bigger overlap between classes on the $x_1$ axis ($\mu_1$ and $\mu_2$ are means of classes $\mathcal{C}_1$ and $\mathcal{C}_2$) than on the $x_2$ axis. The standard deviation (ellipses) should be taken into account when deriving a measure for class separability (Illustration from [Bishop, 1996, p. 107]). . . . . . . . . . . . . . 184

# Acknowledgements

Above all, I would like to thank Mag. Sonja Stappler for supervising this project and providing many helpful advices during the course of the project.
Further, I want to express many thanks to Marijana Zekic, Kate Smith, Andreas Zell and Andrea Beltratti for providing valuable information by answering my inquiries.

# Chapter 1

# Financial time series

## 1.1 Properties of financial time series

The domain of financial time series prediction is a highly complicated task due to following reasons

1. Financial time series often behave nearly like a random-walk process, rendering (from a theoretical point of view) the prediction impossible (Hellström and Holmström [1998]). The predictability of most common financial time series (stock prices, levels of indices) is a controversial issue and has been questioned in scope of the *efficient market hypothesis (EMH).*

2. Financial time series are subject to regime shifting, i.e. statistical properties of the time series are different at different points in time (the process is time-varying, Hellström and Holmström [1998]).

3. Financial time series are usually very noisy, i.e. there is a large amount of random (unpredictable) day-to-day variations (Magdon-Ismail et al. [1998]).

4. In the long run, a new prediction technique becomes a part of the process to be predicted, i.e. it influences the process to be predicted (Hellström and Holmström [1998]).

The first point (predictability issue of financial time series) is discussed below in section 1.2.

The second property of financial time series will be illustrated using the volatil-

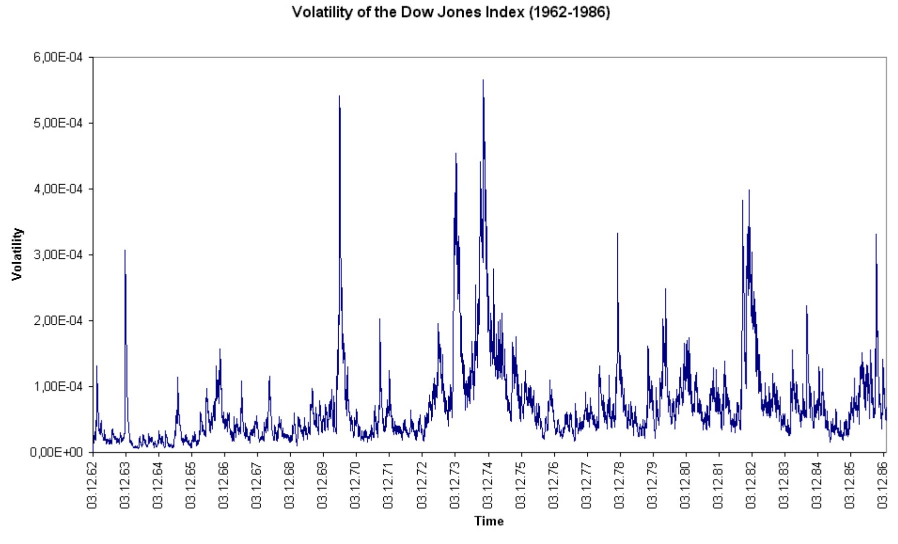



Figure 1.1: Daily volatility of the Dow Jones index from December 3, 1962 to December 31, 1986

ity time series of the Dow Jones index. Consider figure 1.1. It shows the daily volatility of Dow Jones index from 1962 to 1986. The volatility (standard deviation) is an important parameter of a probabilistic distribution. As can be seen from this figure, it changes greatly through time. There are periods, where the index values fluctuates greatly on a single day, and more calm periods. An extreme example is given in figure 1.2, showing the volatility around and on "black monday" (October 19, 1987) stock market crash. And, of course, financial time series are influenced by business cycle.

The third property stems from the fact that financial time series are influenced by events occurring on a certain day (Fair [2000], Donders and Vorst [1996], Fair [2001], Eddelbüttel and McCurdy [1998]). In particular, it was discovered that the volatility of stocks increases before announcement of firm specific news (Donders and Vorst [1996]). As shown in figure 1.3, the volatility of stock prices

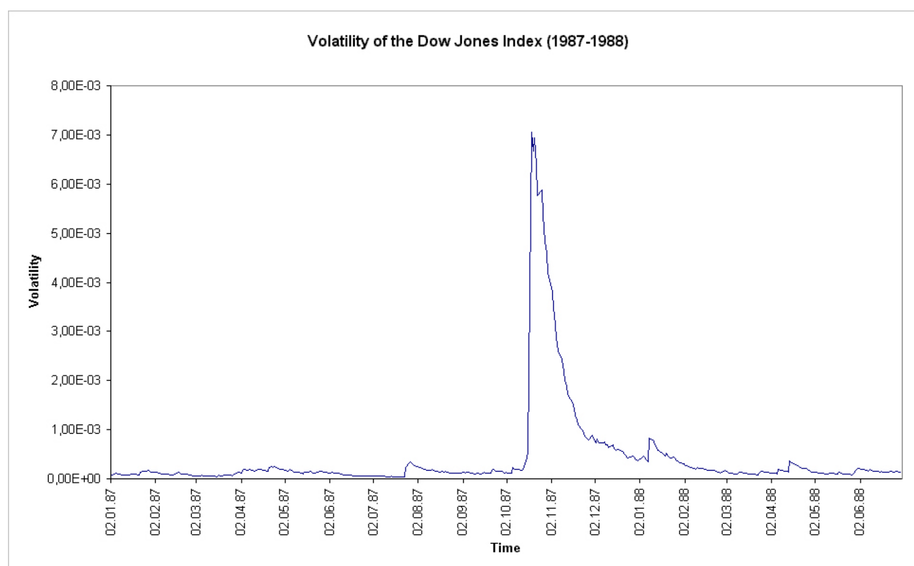



Figure 1.2: Daily volatility of the Dow Jones index from January 2, 1987 to June 6, 1988

increases as the announcement day (the date of announcement is known in advance, but not its content) approaches and decreases sharply after the event (in this case the event is the announcement of firm specific news). In Fair [2000], various types of events are found, whose occurrence coincides with significant changes in S&P 500 index. Such events are fiscal measures (e.g. interest rate changes), employment reports, announcements of macroeconomic news as well as political events. In Fair [2001], the impact of events on stock, bond and exchange rate future prices is examined and again, there are many events that correspond to significant price changes. The events are random and unpredictable and contribute to the noise in the time series.

The noisy nature of financial time series makes it difficult to distinguish a good prediction algorithm from a bad one, since even a random predictor can produce good results (Hellström and Holmström [1998]).

The fourth point will be explained using a thought experiment. Imagine a novel prediction technique is invented, which outperforms all other prediction techniques available. As long as this new technique is used by few market par-

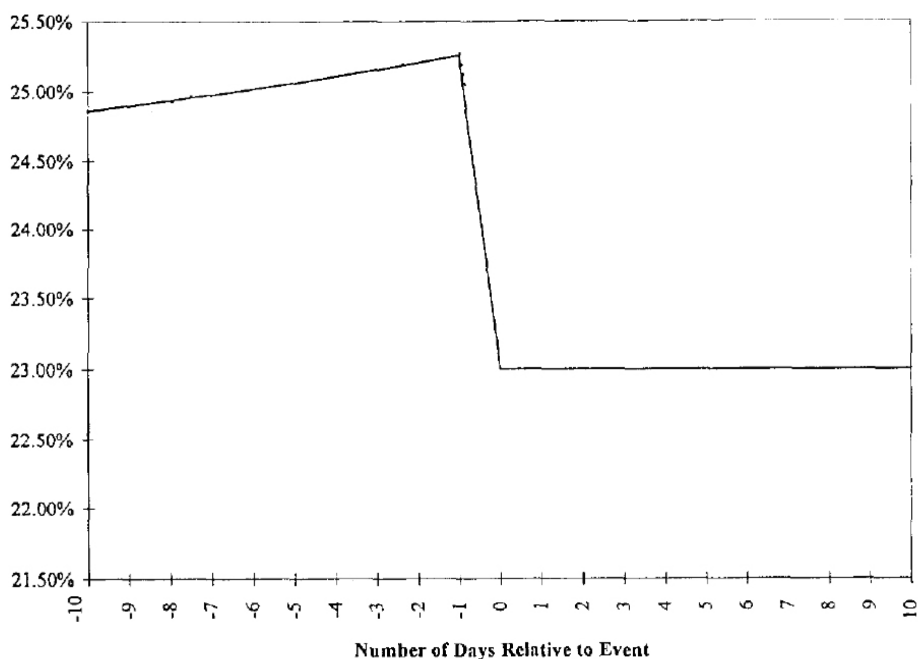



Figure 1.3: Average volatility of stock prices around days on which firm specific news are announced (Donders and Vorst [1996]).

ticipants, its possessors will be able to make extraordinary profits, since they have a more powerful weapon in their prediction techniques arsenal than their competitors. By passage of time, this new technique will become more and more popular among market participants and yielding less and less profit. If everybody employs it, the advantages of the new technique (at least in terms of profit) will vanish. The monopoly on "skill" (better prediction ability) will cease to exist. This argument is stated in Hellström and Holmström [1998] and Swingler [1994]. According to Beltratti [2002], this hypothesis is supported by empirical (simulation-based) results.

## 1.2 Efficient Market Hypothesis

The efficient market hypothesis was developed in 1965 by Fama (Fama [1965], Fama [1970]) and has found broad acceptance (Anthony and Biggs [1995],

Malkiel [1987], Tsibouris and Zeidenberg [1995], White [1988], Lowe and Webb [1991]) in the scientific community (Lawrence et al. [1996]).

The efficient market hypothesis states that the current market price reflects the assimilation of all the information available. This means that given the information, no prediction of future changes in the price can be made. As new information enters the system the unbalanced state is immediately discovered and quickly eliminated by a correct change in market price. Depending on the type of information considered, there exist three forms of EMH (Hellström and Holmström [1998]):

**The weak form** Only past price data is considered. This kind of EMH rules out any form of predictions based on the price data only, since the prices follow a random walk in which successive changes have zero correlation (Hellström and Holmström [1998]).

**The semistrong form** All publicly available information is considered. This includes additional trading information such as volume data (e.g. number of traded stocks) and fundamental data such as profit prognoses and sales forecasts (Hellström and Holmström [1998]).

**The strong form** All information, publicly as well as privately available is considered (Hellström and Holmström [1998]).

In recent years, the EMH became a controversial issue due to many reasons. On one side, it was shown in some studies that excess profits *can* be achieved using only past price data (e.g. Tino et al. [2000]), on the other side it is very difficult to test the strong form due to lack of data.

Another reasonable argument against the EMH deals with the different time perspectives different traders have when they do business. For example, a majority stock owner will react quite differently than a floor trader when a stock suddenly drops in value. These differences in time perspectives will cause anomalies in the market prices even if no new information has entered the scene. It may be possible to identify these situations and actually predict future changes (Hellström and Holmström [1998]).

26 years after the publication of EMH, Fama revised his view concerning market efficiency and wrote (Fama [1991])

> "I take the market efficiency hypothesis to be the simple statement that security prices fully reflect all available information. A precondition for this strong version of the hypothesis is that information and trading costs, the costs of getting prices to reflect information, are always 0... A weaker and economically more sensible version of the efficiency hypothesis says that prices reflect information to the point where the marginal benefits of acting on information (the profits to be made) do not exceed the marginal costs...
> Since there are surely positive information and trading costs, the extreme version of the market efficiency hypothesis is surely false."

Even though the strong form of EMH is said to be false by its developer, this issue remains an interesting research area.
Most often, the arguments in favour of EMH rely on statistical tests showing no predictive power in the tested models and technical indicators. Most arguments against the EMH refer to a time delay between the point when new information enters the system and the point when the information has been assimilated globally and a new equilibrium with a new market price has been reached. Viewed this way, the controversy is only a matter of how the word *immediately* in the EMH definition should be interpreted. It may just be the case that traders in general simply are faster than academics (Hellström and Holmström [1998]).

## 1.3 Data used for financial time series prediction

There are several different types of data which can be assigned to following categories:

**Technical data** This includes such figures as past stock prices, volume, volatility etc. Actually, the term *financial time series* usually refers to time series

of technical data (Hellström and Holmström [1998]).

**Fundamental data** These are data describing current economic activity of the company or companies whose stock prices are to be predicted. Further, fundamental data include information about current market situation as well as macroeconomic parameters (unemployment rate, inflation; Hellström and Holmström [1998]).

The last type of the data, *derived entities*, are produced by transforming and combining technical and/or fundamental data (Hellström and Holmström [1998]).

### 1.3.1 Technical data

Typical technical data involved in financial time series prediction are

- Close value (price of the last performed trade during the day)
- Highest traded price during the day
- Lowest traded price during the day
- Volume (total number of traded stock during the day)

While in most cases, daily data is used for modelling stock price behaviour, data for each individual trade during the day is sometimes available as well. Such data is most often used not for modelling the market, but for determining the right time of an intended trade in real trading (Hellström and Holmström [1998]).

The most obvious choice of entity to predict is the time series of the close values. This approach has some drawbacks, among them (Hellström and Holmström [1998]):

1. The close prices normally vary greatly and make it difficult to create a model for a longer period of time (Hellström and Holmström [1998]).
2. The close prices for different stocks may easily differ over several decades and therefore can not be used as the same type of input in a model (Hellström and Holmström [1998]).

Instead of modelling close prices, returns are the data type of choice in most cases (see section 1.3.3).

### 1.3.2 Fundamental data

Analysis of company's value is usually done by professional market analysts on a regular basis. Their analyses provide a basis for evaluating the true value of company's stock. Fundamental analysts take into consideration following factors (Hellström and Holmström [1998]):

1. The *general state of economy* measured by inflation, interest rate, trade balance etc.

2. The condition of the industry, to which the company belongs measured by

   - Stock price indices (Dow Jones, DAX, FTSE 100, S&P 500 etc)
   - Prices of related commodities such as oil, different metals and currencies.
   - The value of competitors' stocks

3. The condition of the company measured by

   - P/E (price/earnings) ratio (stock price divided by the earning per share during the last 12 months)
   - book value per share (net assets (assets minus liabilities) divided by the total number of shares)
   - Net profit margin (net income divided by total sales)
   - Debt ratio (liabilities divided by total assets)
   - Prognoses of future profits
   - Prognoses of future sales

### 1.3.3 Derived entities

**Returns**

The *one-step returns* $R(t)$ is defined as the relative increase in price since the previous point in the time series $y$:

$$R(t) = \frac{y(t) - y(t-1)}{y(t-1)} \tag{1.1}$$

The *log-return* is defined as

$$R(t) = \log \frac{y(t)}{y(t-1)} \tag{1.2}$$

One-step and log returns are very similar for small changes and are very often

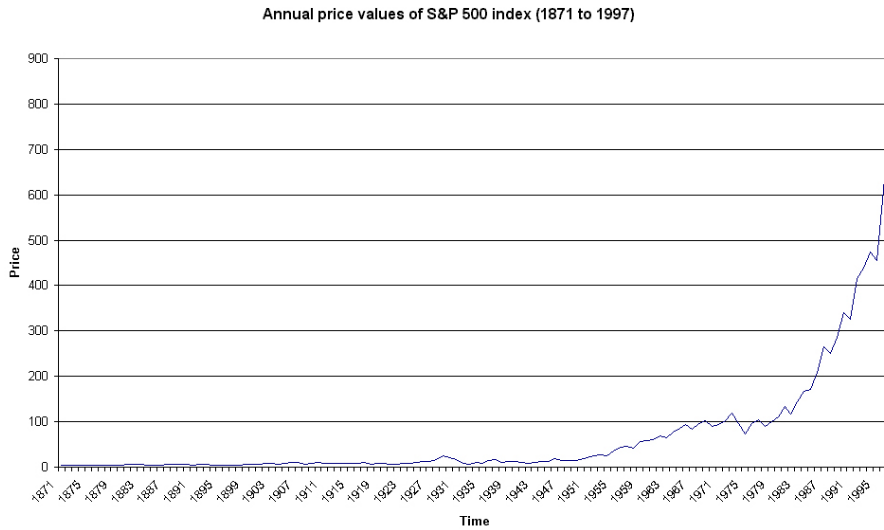



Figure 1.4: Annual values of S&P 500 index (1871 to 1997)

used for financial time series prediction (Hellström and Holmström [1998]) for following reasons:

1. $R(t)$ has a relatively constant range even if data for many years are used as input. The "raw" prices $y$ vary much more and make it difficult to create a valid model for a longer period of time (Hellström and Holmström [1998]). Consider the figures 1.4 and 1.5. They show the values of S&P 100 index

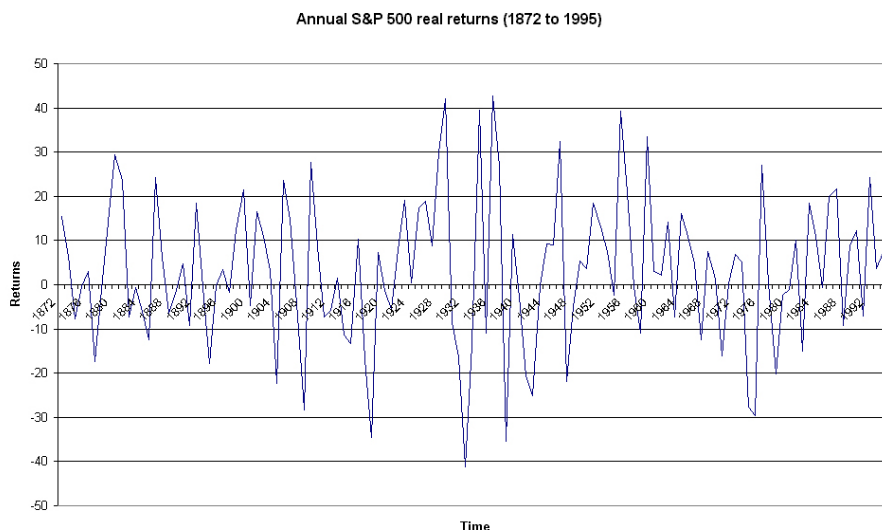



Figure 1.5: Annual S&P 500 real returns (1872 to 1995)

in "raw" form and as returns. The prices (figure 1.4) in the nineteenth century part of the graph are on a completely different scale than those at the end of the twentieth. Returns (figure 1.5) also vary greatly, but they do so within approximately the same boundaries, they remain on the same scale.

2. $R(t)$ for different stocks may also be compared on an equal basis (Hellström and Holmström [1998]).

3. It is easy to evaluate a prediction algorithm for $R(t)$ by computing the prediction accuracy of the sign of $R(t)$. An accuracy above 50 % (more precisely above the mean) indicates that a true prediction has taken place (Hellström and Holmström [1998]).

**Volatility**

Volatility is the synonym for standard deviation of some value (e.g. stock price). Volatility is a measure for risk, but also for profit possibilities. In so-called *delta-*

*neutral trading strategies* (for option contracts[1]), the profit/loss of a trading operation depends not on the price, but on the volatility of the underlying stock price (Tompkins [1994]). Thus, volatility is not only a measure of the risk, but also a tradeable "commodity".

On the other side, it is empirically known that volatility and predictability of financial time series are connected (Siriopoulos et al. [1996]).

### Volume

The increase of traded volume is often viewed as an indication of new information reaching the market. Therefore it may be useful to include the rate of change of volume as an additional input variable (Hellström and Holmström [1998]).

### Turning points

Turning points in a stock price chart can be viewed as positions where equilibrium between demand and supply has been reached. Therefore, they may be viewed as more significant than the data in between, which could be regarded as noise in this context (Hellström and Holmström [1998]).

### Technical indicators

Technical stock analysis deals extensively with derived entities. For a thorough description of the most common technical indicators, see Holmström [1997]. Some examples are (Hellström and Holmström [1998]):

- Relative Strength Index (RSI). The relation between the average upward and downward price changes within a time window of fixed length. The window is normally 14 days backwards (Hellström and Holmström [1998]).
- Moving average (MA). A price rise above a moving average is interpreted as a buy signal, and a fall below it is interpreted as a sell signal (Hellström and Holmström [1998]).

[1] Option contracts give the holder the right, but not the obligation to buy or to sell another stock or commodity (the "underlying" stock) in future at a price fixed at contract signing.

- Moving average convergence divergence (MACD). Two moving averages with different backward windows are used. The difference between the moving averages are also smoothed. Buy and sell signals are generated from the crossings and trends of these filtered price signals (Hellström and Holmström [1998]).

- Relative strength (RS). The concept of relative strength trading involves selecting stocks for trading by their performance compared to the general market, represented by some relevant index (Hellström and Holmström [1998]).

### Other data

Sometimes, other types of data are also used for financial time series prediction, such as artificial data and relative stock performance. Interested reader should refer to Hellström and Holmström [1998].

## 1.4 Classical methods for financial time series processing

Traditional methods of financial time series include following models:

- *Mean model*

$$\hat{y}_t = \mu_y$$

i.e. the predicted value equals to the sample mean of the time series (Nau [2000]).

- *Linear trend model*

$$\hat{y}_t = \alpha + \beta \dot{t}$$

is equivalent to fitting a line to a series of observations such that the residuals are minimized (Nau [2000]).

- *Random walk model*

$$\hat{y}_t = y_{t-1} + \alpha$$

(the estimated value is equivalent to the previous value plus a random difference $\alpha$) is applicable for time series which aren't stationary, but whose first differences are.

- *Geometric random walk model*

$$\log(y_t) = \log(y_{t-1}) + \alpha$$

$$e^x \approx 1 + x \Rightarrow y_t = y_{t-1}e^{\alpha} \Rightarrow y_t \approx y_{t-1}(1 + \alpha)$$

is applicable to time series, which exhibit an exponential irregular growth, but whose logarithmic transformation grows more or less linearly and thus can be approximated by the random walk model.

- *AR, MA, ARMA, ARIMA, ARFIMA* models [2] as well as models based on them (e.g. including some specific transformations aiming at removing a seasonal trend) are variants of the ARIMA model presented in section 1.4.1.

- The *ARCH* model family (ARCH , GARCH, I-GARCH, GARCH-M) [3] is used to model time series whose variance changes in time (Ruppert [2001a]).

- *Capital Asset Pricing Model* (CAPM) models the return for individual securities or portfolios (Kerr [1997]).

- The *Black and Scholes* model is used for modelling prices of options taking into consideration the specific properties of them (Tompkins [1994]).

### 1.4.1 ARIMA process

#### Notation

$\varepsilon_t$ White noise

[2] ARIMA = Autoregressive Integrative Moving Average, ARFIMA = Autoregressive Fractionally Integrative Moving Average

[3] ARCH = Autoregressive Conditional Heteroscedasticity, GARCH = Generalized Autoregressive Conditional Heteroscedasticity

$\rho$ Autocorrelation function

$\gamma(h)$ Autocovariance function (covariance between $X_t$ and $X_{t+h}$)

$\hat{h}(h) = \frac{1}{n}\sum_{j=1}^{n-h}(y_{j+h} - \bar{y})(y_j - \bar{y})$ Estimate of the autocovariance function

$\hat{\rho}(h) = \frac{\hat{\gamma}(h)}{\hat{\gamma}(0)}, \hat{\rho}(h) = \hat{\rho}(-h)$ Estimate of the autocorrelation function (SACF, sample autocorrelation function)

$\mathrm{B}^k\, y_t = y_{t-k}$ Backwards operator

$\Delta y_t = y_t - \mathrm{B}\, y_t, \Delta y_t = (1 - \mathrm{B})y_t = y_t - y_{t-1}$ Differencing operator

$e$ Vector, representing residuals

$s_u^2$ Sample variance of the disturbance term

$n$ Number of elements of a time series (sample size)

**AR and MA processes**

A variable $y_t$ is autoregressive of order $p$ ($AR(p)$) if it is a function of its past values (Glewwe [2000]):

$$\mathrm{AR}(p) : y_t = \mu + \left(\sum_{i=1}^{p} \gamma_i y_{t-1}\right) + \varepsilon_t \tag{1.3}$$

where $\varepsilon_t$ is white noise, i.e. a random number from a distribution with following properties:

$$\mathrm{E}(\varepsilon_t) = 0 \tag{1.4}$$

$$\mathrm{E}(\varepsilon_t^2) = \rho_e^2 \tag{1.5}$$

$$\mathrm{Cov}(\varepsilon_t, \varepsilon_s) = 0, s \neq t \tag{1.6}$$

Cov means *Covariance* and is a measure of association between two variables (Dougherty [1992]):

$$\mathrm{Cov}(x, y) = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y}) \tag{1.7}$$

Hence, white noise are randomly distributed real numbers with zero mean and no association between numbers drawn at different points of time. Figure 1.6

shows some examples of AR processes. An example of fitting an AR(1) to a real time series can be found in appendix A. A *moving average* (MA) process

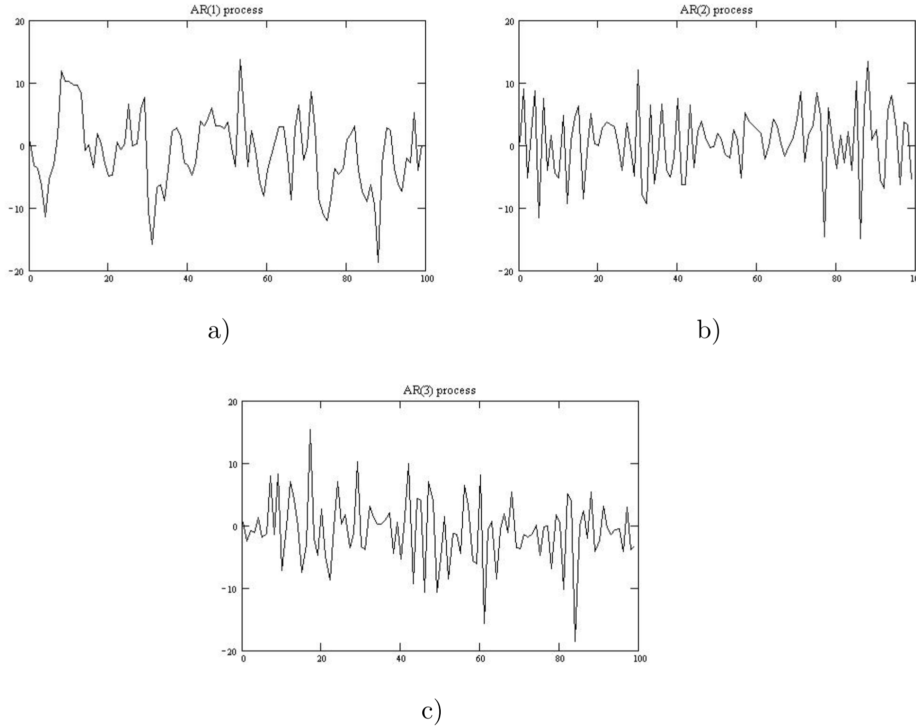



Figure 1.6: Examples of AR processes with different parameters (in all cases there are 100 observations, $x_0 = 0.5$ and $\mu = 0.2$). Figure a) shows an $AR(1)$ process with $\gamma = 0.5$. Figure b) shows an $AR(2)$ process with $\gamma_1 = -0.25$, $\gamma_2 = -0.25$. Figure c) is an $AR(3)$ process with $\gamma_1$=-0.3, $\gamma_2 = -0.3$, $\gamma_2$=-0.3.

is defined as

$$\mathrm{MA}(q) : y_t = \mu + \varepsilon_t - \sum_{i=1}^{q} \theta_i \varepsilon_{t-i} \tag{1.8}$$

Since every MA process can be rewritten as an AR process (Glewwe [2000]), the diagram showing example processes is omitted here.

**ARIMA process**

An ARMA (autoregressive moving average) model is defined as:

$$\mathrm{ARMA}(p, q) : y_t = \mu + \left( \sum_{i=1}^{p} \gamma_i y_{t-1} \right) + \varepsilon_t + \left( \sum_{j=1}^{q} \theta_j \varepsilon_{t-j} \right) \tag{1.9}$$

In practice, one applies the ARMA process not to the time series, but to transformed time series. It is often the case, that the time series of differences is stationary in spite of the non-stationarity of the underlying process. Stationary time series can be well estimated by the ARMA model. That leads to the definition of the ARIMA model:

$$\Delta^d y_i \text{is ARMA}(p,q) \text{ process} \rightarrow y_t \text{is ARIMA}(p,d,q) process$$

Thus, an ARIMA$(p,d,q)$ process models the stationary differences of the order d of the time series $y_t$ using the ARMA$(p,q)$ process. For the forecasting purposes the usage of ARIMA model can be summarized by the flowchart shown in figure 1.7. In the first step, the differencing operator is applied to the time series until

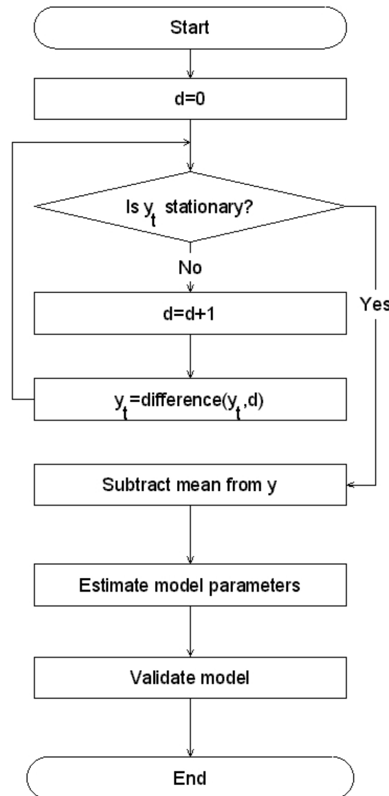



Figure 1.7: Preparation of the time series for usage with the ARIMA model

it becomes stationary. Normally, the value of $d$ does not exceed 2 (Ruppert [2001b]). The determination whether $y_t$ is stationary or not is performed by

visualizing the $SACF$ of $y_t$ (this diagram is called correlogram). Several authors suggest this technique (Pollock [1992], Robinson [1999], Ruppert [2001b]). The SACF converges to zero, if the process is a stationary one. The SACF of a non-stationary process decays to zero much more slowly (see figure 1.8). After

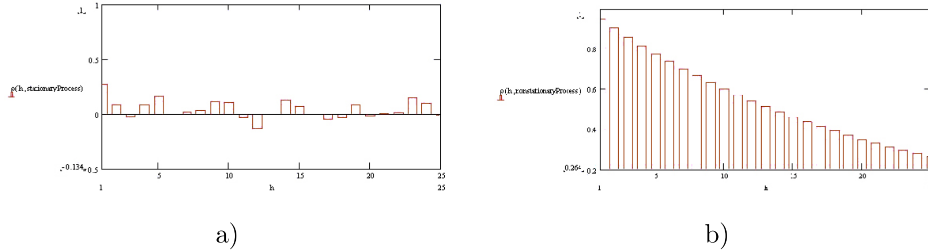



Figure 1.8: Visualisation of the SACFs of a stationary (a)) and a non-stationary process (b))

the $d$ value is determined, any non-zero mean is removed from the time series (the mean is calculated and subtracted from each element of the time series). Afterwards, the parameters $p$ and $q$ are estimated. This is usually done by fitting models with different $p$ and $q$ parameters to the test data and choosing one that matches the time series best. Due to the high number of different approaches to this task, the detailed explanation of this step is omitted here. Most textbooks use a computer program for demonstrating the use of ARMA model in practice. So does the author in appendix C.

If after the validation (last step) one figures out that the model fits the data well enough, it is used for forecasting purposes. In order to explain how forecasting is performed with the ARIMA model, we need to add some formal definitions. The ARMA model can - apart from the definition given above - be written in the following form (Ruppert [2001b]):

$$\left(1 - \sum_{i=1}^{p} \gamma_i \,\mathrm{B}^i \, y_i\right)(y_t - \mu) = \left(1 - \sum_{j=1}^{q} \theta_j \,\mathrm{B}^j \, y_j\right) \varepsilon_t \tag{1.10}$$

If

$$\gamma(B) = 1 - \sum_{i=1}^{p} \gamma_i \,\mathrm{B}^i \quad \text{and} \quad \theta(B) = 1 - \sum_{j=1}^{q} \theta_j \,\mathrm{B}^j \tag{1.11}$$

then ARMA process can further be defined as (Borchers [2001])

$$\gamma(B)y_t = \theta(B)\varepsilon_t \tag{1.12}$$

Note that this definition is valid only if the mean value has previously been subtracted from the time series. Now, we define the so-called transfer function

$$\psi(B) = \frac{\theta(B)}{\phi(B)} \tag{1.13}$$

and rewrite the definition of the ARMA process as (Borchers [2001])

$$y_t = \psi(B)a_n \tag{1.14}$$

The transfer function can be expanded as

$$\psi = 1 + \sum_{i=1}^{\infty} \psi_i \, \mathrm{B}^i \tag{1.15}$$

The coefficients $\psi_i$ can be obtained using the Taylor series expansion (see appendix E). Predictions of time series elements are given by

$$\hat{y}_t = \sum_{j=t}^{\infty} \psi_j \varepsilon_{n+t-j} \tag{1.16}$$

The random error associated with the forecast is defined as

$$e_n(t) = \varepsilon_{n+t} + \sum_{i=1}^{t-1} \psi_i \varepsilon_{n+t-i} \tag{1.17}$$

In order to calculate the forecasts, we have to calculate the series $\varepsilon$ corresponding to the time series $y$. Since

$$e_n(1) = \varepsilon_{n+1} \quad a_n \text{ is equal to } \quad a_n = y_n - \hat{y}_{n-1}(1) \tag{1.18}$$

Now it is time to give an overview about the forecast procedure and that is done with a diagram shown in figure 1.9. Written in the mixed form, the ARIMA model looks like

$$\left(1 - \sum_{i=1}^{p} \gamma_i \, \mathrm{B}^i \, y_i\right) \left(\prod_{k=1}^{d} (1 - \mathrm{B})\right) = \left(1 - \sum_{j=1}^{q} \theta_j \, \mathrm{B}^j \, y_j\right) \varepsilon_t \tag{1.19}$$

Note that - contrary to the ARMA model - this form takes into consideration that the ARIMA model is an ARMA model operating on differences of the order $d$ of the original values, hence there is a need to add the term

$$\left(\prod_{k=1}^{d} (1 - B)\right)$$

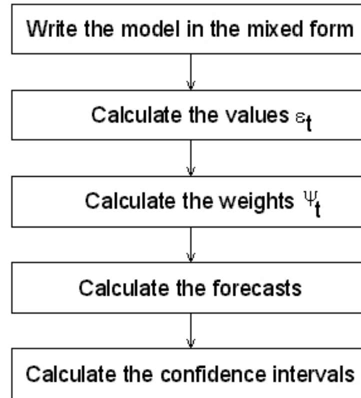



Figure 1.9: The forecasting process

to the model definition. Again, the mean of the time series has been subtracted from the original values, thus there is only $y_t$ term on the left side of the equation, not $(y_t - \mu)$. Written in this form, we can derive the actual values of $\gamma$ and $\theta$ with respect to the $d$ parameter of the ARIMA model. Having these actual values, we can calculate the lag 1 predictions:

$$\hat{y}_{n-1}(1) = \left(\sum_i \gamma_i y_{n-i}\right) + \left(\sum_j \theta_i \varepsilon_{n-j}\right) \tag{1.20}$$

Be aware that here we are using the actual values of $\gamma$ and $\theta$ obtained from the mixed form equation of the ARIMA model. Using this information, we obtain the values of $\varepsilon$ for the part of time series we know. If we have a time series

$$Z = z_1, z_2, \ldots, z_n$$

and we want to use first 10 elements of the time series to forecast the values after the 10th element, we must calculate the $\varepsilon$ value for the first 10 elements (i.e. for the "training" set of the ARIMA model). Next, the transfer function

$$\psi(B) = \frac{\theta(B)}{\phi(B)}$$

has to be expanded to obtain the $\psi_i$ weights. Now we come to the crucial point - namely the calculation of the forecasts themselves. They are defined as:

$$\hat{y}_t = \left(\sum_i \gamma_i y_{t-i}\right) + \left(\sum_j \theta_i \varepsilon_t\right) \tag{1.21}$$

Having computed the forecasts, we have to calculate the confidence intervals of them. First we calculate the variances for each of the forecasts:

$$t = 1 \Rightarrow \sigma_t^2 = \sigma_\varepsilon^2 \tag{1.22}$$

$$t > 1 \Rightarrow \sigma_t^2 = \sigma_\varepsilon^2 \left(1 + \sum_{i=1}^{t-1} \psi_i^2\right) \tag{1.23}$$

$\sigma_\varepsilon$ is the standard deviation of the $\varepsilon$ values. The last step is the calculation of confidence intervals:

$$\text{Lower confidence interval} : \hat{y}_t - Z\sigma_t \tag{1.24}$$

$$\text{Upper confidence interval} : \hat{y}_t + Z\sigma_t \tag{1.25}$$

The value $Z$ depends on the confidence interval we use (e.g. 1.96 for a 95 % two-tailed confidence interval). In this way, we can obtain information, which points lie within a particular confidence interval (the confidence interval is the range we expect the actual values to lie, University of Derby in Austria; see appendix F). An example of forecasting using ARIMA process is given in appendix D.

### 1.4.2 ARCH models

#### Motivation

The ARIMA models presented in section 1.4.1 have one severe drawback: they assume that the volatility [4] of the variable being modelled (e.g. stock price) is constant over time. In many cases this is not true. Large differences (of either sign) tend to be followed by large differences. In other words, the volatility of asset returns appears to be serially correlated (Campbell et al. [1997]). Figure 1.10 shows monthly excess returns on the CRSP value-weighted stock index over the period from 1926 to 1994. The individual monthly returns vary wildly, but they do so within a range which itself changes slowly over time. The range for returns is very wide in the 1930s, for example, and much narrower in the 1950s and 1960s ([Campbell et al., 1997, p. 482]).
ARCH and related models were developed in order to capture this property of financial time series.

[4] Volatility is the synonym for standard deviation.

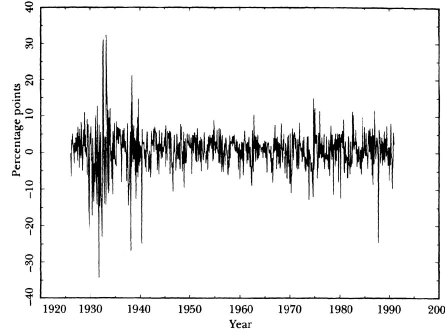



Figure 1.10: Monthly Excess Log US Stock Returns, 1926 to 1994 ([Campbell et al., 1997, p. 482])

### ARCH model

The ARCH (Autoregressive Conditional Heteroscedasticity)[5] process is defined as

$$\text{ARCH(q)}:\ y_t = \sigma_t \epsilon_t \tag{1.26}$$

$$\sigma_t = \sqrt{\alpha_0 + \sum_{i=1}^{q} \alpha_i y_{t-i}^2} \tag{1.27}$$

where $\sigma_t$ is the conditional standard deviation of $y_t$ given the past values of this process. The ARCH(q) process is uncorrelated and has a constant mean, a constant unconditional variance ($\alpha_0$), but its conditional variance is nonconstant. This model has a simple intuitive interpretation as a model for volatility clustering: large values of past squared returns ($y_{t-i}^2$) give rise to a large current volatility (Martin [1998]).

The ARCH(q) model is a special case of the more general GARCH(p,q) model defined as (GARCH means "Generalised ARCH")

$$\text{GARCH(p,q)}:\ y_t = \sigma_t \epsilon_t \tag{1.28}$$

$$\sigma_t = \sqrt{\alpha_0 + \sum_{i=1}^{p} \alpha_i y_{t-i}^2 + \sum_{j=1}^{q} \beta_j \sigma_{t-j}^2} \tag{1.29}$$

[5]This section is based upon Ruppert [2001a].

In this model, the volatility today depends upon the volatilities for the previous $q$ days and upon the squared returns for the previous $p$ days.

A long and vigorous line of research followed the basic contributions of Engle and Bollerslev (developers of ARCH and GARCH model respectively), leading to a number of variants of the GARCH$(p,q)$ model. These include power GARCH (PGARCH) models, exponential GARCH (EGARCH) models, threshold GARCH (TGARCH) models and other models that incorporate so-called *Leverage effects*. Leverage terms allow a more realistic modelling of the observed asymmetric behaviour of returns according to which a "good-news" price increase yields lower subsequent volatility, while a "bad-news" decrease in price yields a subsequent increase in volatility. It is also worth mentioning two-component GARCH models which reflect differing short term and long term volatility dynamics, and GARCH-in-the-mean (GARCH-M) models which allow the mean value of returns to depend upon volatility (Martin [1998]).

# Chapter 2

# Neural networks

## 2.1 Introduction

### 2.1.1 What are neural networks?

Neural network learning methods provide a robust approach to approximating real-valued, discrete-valued and vector-valued target functions. For certain types of problems, such as learning to interpret complex real-world sensor data, artificial neural networks are among the most effective learning methods currently known ([Mitchell, 1997, p. 81]).

The study of artificial neural networks (ANNs) has been inspired in part by the observation that biological learning systems are built of very complex webs of interconnected neurons. In rough analogy, artificial neural networks are built out of a densely interconnected set of simple units, where each unit takes a number of real-valued inputs (possibly the outputs of other units) and produces a single real-valued output, which may become input to other units ([Mitchell, 1997, p. 82]).

To develop a feel for this analogy, let us consider a few facts from neurobiology. The human brain, for example, is estimated to contain a densely interconnected network of approximately $10^{11}$ neurons, each connected, on average, to $10^4$ others. Neuron activity is typically excited or inhibited through connections to

other neurons. The fastest neuron switching times are known to be on the order of $10^{-3}$ seconds, quite slow compared to computer switching speeds of $10^{-10}$ seconds. Yet humans are able to make surprisingly complex decisions, surprisingly quickly. For example, it requires approximately $10^{-1}$ seconds to visually recognize one's mother. Notice the sequence of neuron firings that can take place during this $10^{-1}$-second interval cannot possibly be longer than a few hundred steps, given that the information-processing abilities of biological neural systems must follow from highly parallel processes operating on representations that are distributed over many neurons. One motivation for ANN systems is to capture this kind of highly parallel computation based on distributed representations. Most ANN software runs on sequential machines emulating distributed processes, although faster versions of the algorithms have also been implemented on highly parallel machines and on specialized hardware designed specifically for ANN applications ([Mitchell, 1997, p. 82]).

While ANNs are loosely motivated by biological neural systems, there are many complexities to biological neural systems that are not modelled by ANNs, and many features of ANNs are known to be inconsistent with biological systems ([Mitchell, 1997, p. 82]). The ANN related research can be divided into two directions:

- research targeting at exploring the properties of biological systems by means of neural networks (computational neuroscience) and
- research targeting on development of systems capable to approximate complex functions efficiently and independent of whether they "mirror" biological ones or not.

By "neural network" in scope of this work the author always refers to the second type of neural networks, if not stated otherwise.

### 2.1.2 Basic definitions

The structure of a neural network of most commonly used type is schematically shown in figure 2.1. It consists of several layers of processing units (also termed

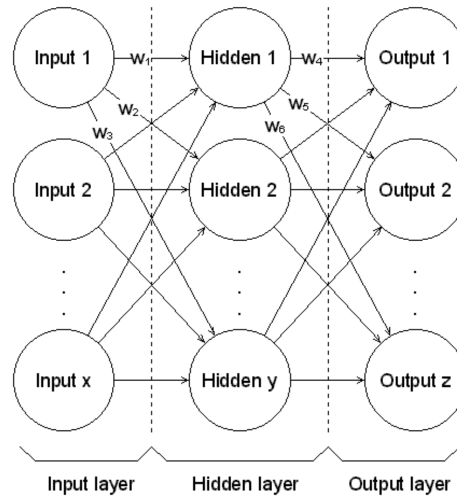



Figure 2.1: Basic structure of a multi-layer perceptron

neurons, nodes). The input values (input data) are fed to the neurons in the so-called *input layer* in the left part of figure 2.1. The input values are processed (the data processing in the neurons is discussed later in this chapter) within the individual neurons of the input layer and then the output values of these neurons are forwarded to the neurons in the *hidden layer*. The arrows indicate connections from the input nodes to hidden nodes, along which the output values of the input nodes are passed on to the hidden nodes. These values obtained as inputs by the hidden nodes are again processed within them and passed on to either the output layer or to the next hidden layer (there can be more than one hidden layer).

Each connection has an associated parameter indicating the strength of this connection, the so-called *weight*. By changing the weights in a specific manner, the network can "learn" to map patterns presented at the input layer to target values on the output layer. The description of the procedure, by means of which this weight adaptation is performed, is called *learning* or *training algorithm*.

Sometimes, so-called *bias units* (also called bias parameters, thresholds) are also present in the neural network (see figure 2.2). These are neurons with the property that they always produce a +1 at the output. The constant output value

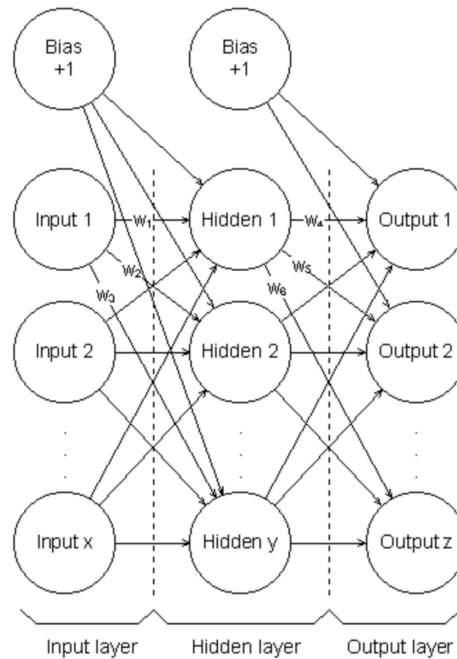



Figure 2.2: Neural network with bias units.

of the bias node is described in most cases by the letter $\theta$.

Usually, the data available for training the network is divided in (at least) two non-overlapping parts: the so-called *training* and *testing set*. The commonly larger training set is used to "teach" the network the desired target function. Then the network is applied to data in the test set in order to test its *generalization ability*, i.e. the ability to derive correct conclusions about the population properties of the data from the sample properties of the training set (e.g. if a network has to learn a sine function, it should produce correct results for all real numbers and not only for those in the training set). If the network is not able to generalize, but instead learns the individual properties of the training patterns without recognizing the general features of the data (i.e. produces correct results for training patterns, but has a high error rate in the test set), it is said to be *overfitted* or to be subject to *overfitting*.

### 2.1.3 Properties of neural networks

ANN learning is well-suited to problems in which the training data corresponds to noisy, complex sensor data, such as inputs from cameras and microphones. It is also applicable to problems for which more symbolic representations are often used, such as the decision tree learning tasks. In these cases ANN and decision tree learning often produce results of comparable accuracy (Mitchell [1997]). The backpropagation algorithm is the most commonly used ANN learning technique. It is appropriate for problems with the following characteristics (Mitchell [1997]):

- *Instances are represented by many attribute value pairs.* The target function to be learned is defined over instances that can be described by a vector of predefined features, such as the pixel values. These input attributes may be highly correlated or independent of one another. Input values can be any real values.

- *The target function output may be discrete-valued, real-valued, or a vector of several real- or discrete-valued attributes.*

- *The training examples may contain errors.* ANN learning methods are quite robust to noise in the training data.

- *Long training times are acceptable.* Network training algorithms typically require longer training times than, say, decision tree learning algorithms. Training times can range from a few seconds to many hours, depending on factors such as the number of weights in the network, the number of training examples considered, and the settings of various learning algorithm parameters.

- *Fast evaluation of the learned target function may be required.* Although ANN learning times are relatively long, evaluating the learned network, in order to apply it to a subsequent instance, is typically very fast.

- *The ability of humans to understand the learned target function is not important.* The weights learned by neural networks are often difficult for

humans to interpret. Learned neural networks are less easily communicated to humans than learned rules.

### 2.1.4 Multi-layer perceptron

There exists a huge variety of different ANN types (ANN topologies), but the most commonly used one is the *multi-layer perceptron* (MLP).

In the 1960es there was a great euphoria in the scientific community about ANN based systems that were promised to deliver breakthroughs in many fields. Single-layer neural networks such as ADALINE[1] were used widely, eg in the domain of signal processing. This euphoria was given an end by the publication of Minsky and Papert (Minsky and Papert [1969]), who showed that the ANNs used at that time were not capable of approximating target functions with certain properties (target functions that are not linearly separable such as the "exclusive or" (XOR) function). In the 1970es only a small amount of research was devoted to ANNs. In the mid-1980s, the ANNs were "revived" by invention of the *error backpropagation* (EBP) learning algorithm in combination with multi-layer networks. As can be seen in figure 2.1, a MLP consists of at least three layers: input layer, one or many hidden layers and output layer. The individual nodes are connected by links where each link has a certain *weight* ($w_1, w_2, w_3, w_4, w_5, w_6$ in the figure, note that contrary to the figure, *each* link has a weight). Each node takes multiple values as input, processes them, and produces an output, which can be "forwarded" to other nodes. Given a node $j$,

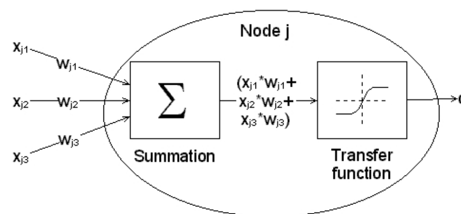



Figure 2.3: Data processing within an ANN node

[1]ADALINE = ADAptive LInear NEuron

its output is equal to

$$o_j = transfer\left(\sum(x_{ji}w_{ji})\right) \tag{2.1}$$

where $o_j$ is the output of node $j$, $x_{ji}$ the $i$th input to unit $j$, $w_{ji}$ the weight associated with $i$th input to unit $j$ and $transfer$ a transfer function discussed later in this section. Figure 2.3 visualizes the data processing that takes place within a node of an ANN. Note that a neuron may have an arbitrary number of inputs, but only one output. By changing the weights of the links connecting individual nodes, the ANN can be adjusted for approximating a certain function. An example is given further in this section. The non-linear transfer function

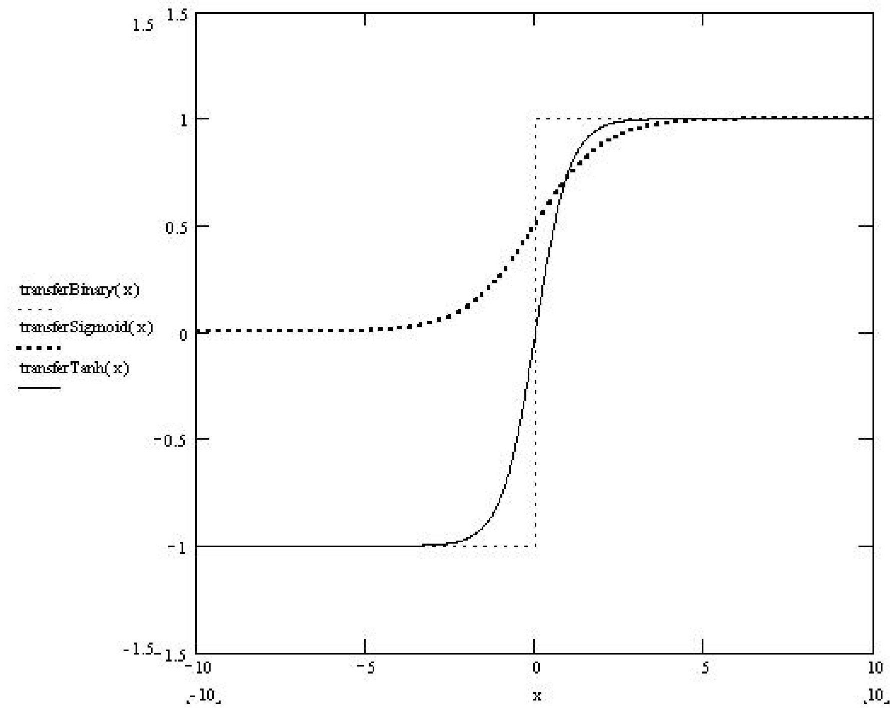



Figure 2.4: Binary, sigmoid and tangens hyperbolicus transfer functions

$transfer$ is responsible for transferring the weighted sum of inputs to some value that is given to the next node. There are several types of transfer functions, among which

- Binary transfer function

$$transfer(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ -1 & \text{otherwise} \end{cases} \tag{2.2}$$

and its variants (eg 0 and 1 instead of 1 and -1) can be used sometimes, however in most cases its use is not appropriate due to the fact that it is not a smooth function.

- Sigmoid transfer function

$$transfer(x) = \frac{1}{1 + e^{-x}} \tag{2.3}$$

is the function used most often in MLPs. Note that the return value of this function lies in the interval [0,1], so this function cannot be used in ANNs that approximate functions, which can also take on negative values (e.g. returns time series).

The reason for the popularity of this transfer function lies in the fact that its first derivative (which is needed for training the ANN), is a very simple expression:

$$transfer'(x) = \left(\frac{1}{1 + e^{-x}}\right)'$$

$$transfer'(x) = \left(-1 \cdot (1 + e^{-x})^{-2}\right) \cdot (1 + e^{-x})'$$

$$transfer'(x) = \left(-1 \cdot (1 + e^{-x})^{-2}\right) \cdot (e^{-x}) \cdot (-1)$$

$$\Rightarrow transfer'(x) = \frac{e^{-x}}{(1 + e^{-x})^2}$$

$$transfer'(x) = transfer(x) \cdot (1 - transfer(x))$$

$$\text{since} \quad transfer(x) \cdot (1 - transfer(x)) = \left(\frac{1}{1 + e^{-x}}\right) \cdot \left(1 - \frac{1}{1 + e^{-x}}\right)$$

$$transfer(x) \cdot (1 - transfer(x)) = \frac{e^{-x}}{(1 + e^{-x})^2}$$

- Tangens hyperbolicus (tanh) transfer function

$$transfer(x) = tanh(x) = \frac{sinh(x)}{cosh(x)} = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{2.4}$$

has the advantage of having an output interval of [-1,1], thus it can be used in ANNs that need to approximate functions that can take on negative values (eg stock index differences).

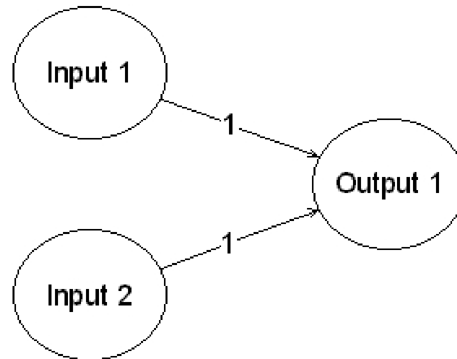



Figure 2.5: ANN approximating the OR function

Consider the simple ANN shown in figure 2.5. It approximates the OR function. Assume the transfer function of the output node is binary and the values the input nodes can take on are -1 (false) and 1 (true). In table 2.1 it is shown how the ANN yields the correct results. The theoretical ability of ANNs of certain

| **Input 1** | **Input 2** | **Output 1** |
|---|---|---|
| -1 | -1 | $(-1) \cdot 1 + (-1) \cdot 1 = -2, transfer(-2) = -1$ |
| -1 | 1 | $(-1) \cdot 1 + 1 \cdot 1 = 0, transfer(0) = 1$ |
| 1 | -1 | $1 \cdot 1 + (-1) \cdot 1 = -2, transfer(0) = 1$ |
| 1 | 1 | $1 \cdot 1 + 1 \cdot 1 = 2, transfer(2) = 1$ |

Table 2.1: Reaction of the network approximating the OR function to different inputs

types to represent functions of arbitrary complexity has been proven by means of the *Kolmogorov's theorem* that is described in appendix G.

### 2.1.5 Learning algorithms

Contrary to such a simple case as the OR function is, usually the weights of the ANN must be adjusted using some learning algorithm in order for the ANN to be able to approximate the target function with a sufficient precision. Two of these learning algorithms are presented in following sections.

### 2.1.6 Stochastic gradient descent backpropagation learning algorithm

Usually, the term "neural network" refers to a MLP trained with this learning algorithm, often called "backpropagation" or "error-backpropagation" (EBP). Assume an ANN uses the following error function

$$E(\vec{w}) = \frac{1}{2} \sum_{d \in D} \sum_{k \in outputs} (t_{kd} - o_{kd})^2 \tag{2.5}$$

where $o_{kd}$ is the output value produced by output neuron $k$, $t_{kd}$ the desired (correct) value this neuron should produce and $D$ denotes the set of all training patterns, i.e. $E(\vec{w})$ is the sum of prediction errors for all training examples. Prediction errors of individual training examples are in turn equal to the sum of the differences between output values produced by the ANN and the desired (correct) values. $\vec{w}$ is the vector containing the weights of the ANN.

The goal of a learning algorithm is to minimize $E(\vec{w})$ for a particular set of training examples. There are several ways to achieve this, one of them being the so-called *gradient descent* method. Basically, it works in the following way (Schraudolph and Cummins [2002]):

1. Choose some (random) initial values for the model parameters.
2. Calculate the gradient $G$ of the error function with respect to each model parameter.
3. Change the model parameters so that we move a short distance in the direction of the greatest rate of decrease of the error, i.e. , in the direction of $-G$.
4. Repeat steps 2 and 3 until G gets close to zero.

*Gradient* $G = \nabla f(x)$ of function $f$ is the vector of first partial derivatives (The Numerical Algorithms Group Ltd)

$$\nabla f(x) = \left( \frac{\partial f(x)}{\partial x_1}, \frac{\partial f(x)}{\partial x_2}, \ldots, \frac{\partial f(x)}{\partial x_n} \right) \tag{2.6}$$

In our case, $G = \nabla E(\vec{w})$ (i.e. the derivative of the error function $E$ with respect to the weight vector $\vec{w}$). When interpreted as a vector in weight space,

the gradient specifies the direction that produces the steepest increase in $E$ ([Mitchell, 1997, p. 91]). The negative of this vector therefore gives the direction of steepest decrease.

Consider the figure 2.6. It shows the behaviour of $E$ with respect to one weight $w$. In order to decrease the value of error function $E$, we always must move in the reverse direction of the gradient (slope). If the gradient of $E$ is negative, we must increase $w$ to move "forward" towards the minimum. If $E$ is positive, we must move "backwards" to the minimum. Note that this figure shows the relation between error function and weights only schematically. It shows the relation between $E$ and only one particular weight $w$ (in most neural networks there are many weights). By repeating this over and over, we move "downhill" in $E$ until

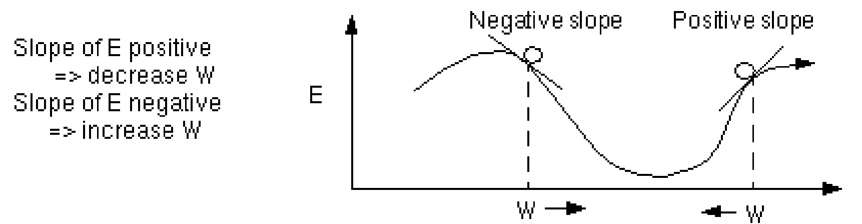



Figure 2.6: Basic principle of gradient descent (Schraudolph and Cummins [2002]).

we reach a minimum, where $\nabla E(\vec{w}) = 0$, so that no further progress is possible (figure 2.7). Having this in mind, we will now explore the *Stochastic gradient descent backpropagation* (error backpropagation) learning algorithm shown in figure 2.8. First, a neural network is created and initialized (weights are set to small random numbers). Then, until the termination condition (eg the mean squared error of the ANN is less than a certain error threshold) is met, all training examples are "taught" the ANN. Inputs of each training example are fed to the ANN, and processed from the input layer, over the hidden layer(s) to the output layer. In this way, vector $o$ of output values produced by the ANN is obtained (step 3a).

In the next step, the weights of the ANN must be adjusted. Basically, this

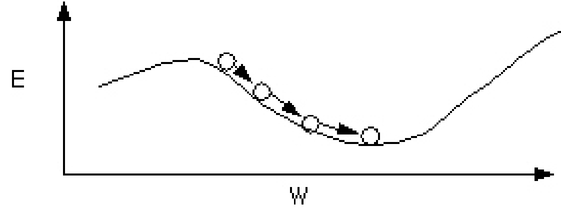



Figure 2.7: Gradient descent: movement to the minimum (Schraudolph and Cummins [2002]).

happens by "moving" the weight in the direction of steepest descent of the error function. This happens by adding to each individual weight the value $\Delta w = \eta\delta_j x_{ji}$. The explanation of this term follows.

In the following explanation, the error function

$$E_d(\vec{w}) = \frac{1}{2}\sum_{k\in outputs}(t_k - o_k)^2 \tag{2.7}$$

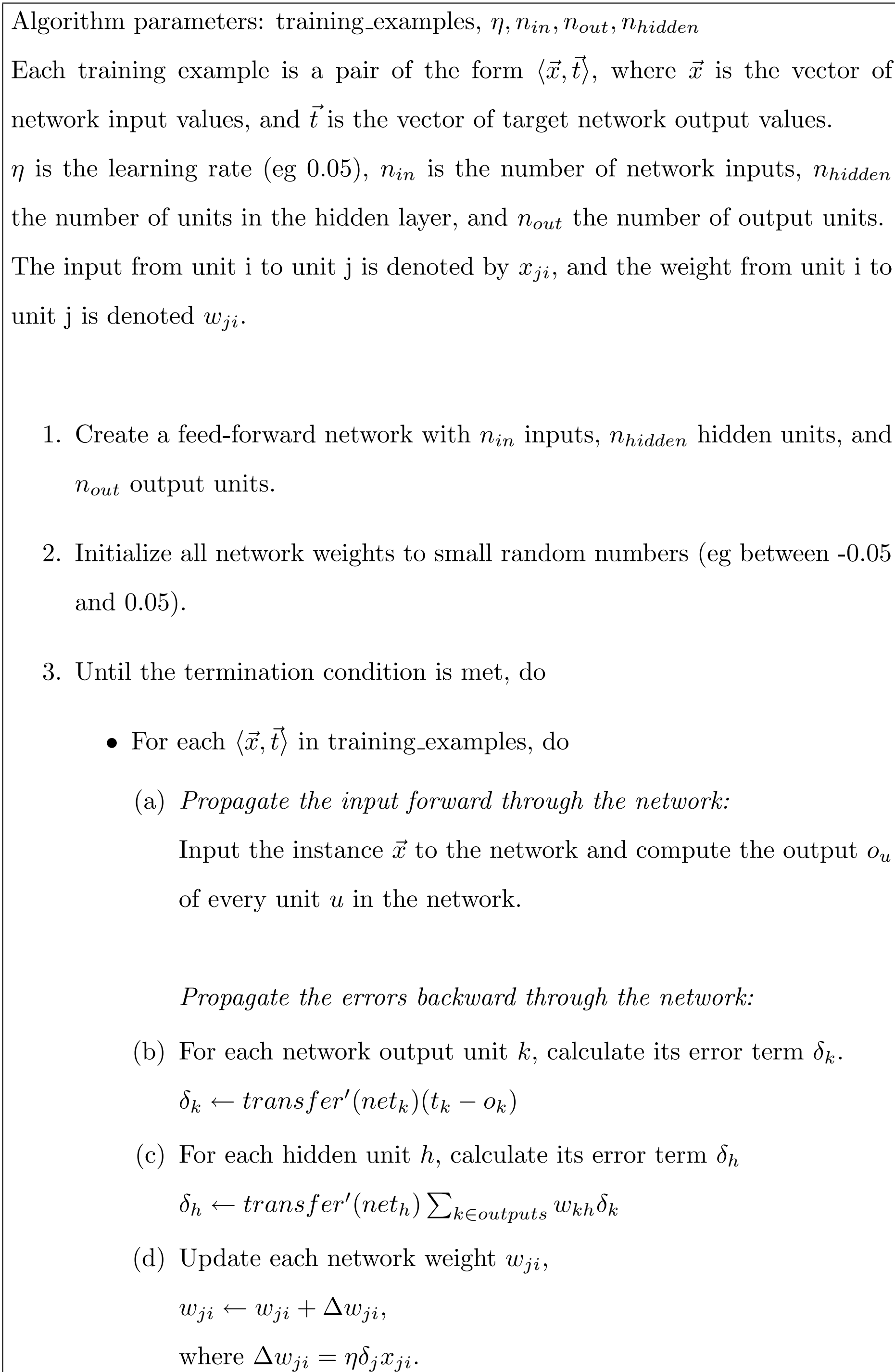
will be used. It is the error of the ANN for the training example $d$.

As mentioned above, we are interested in reducing the error $E$ of the ANN in terms of equation 2.5 by reducing $E_d$ for all $d$ (training examples). Hence, the weight update value $\Delta w$ must "move" the weight in the direction of steepest descent of the error function $E_d$. And this value is equal to the steepest descent of the error function $E_d$ with respect to the weight, or the partial derivative of $E_d$ with respect to the weight. So,

$$\Delta w = -\eta \cdot \frac{\partial E_d}{\partial w_{ji}} \tag{2.8}$$

The term $\eta$ is the *learning rate* that determines the size of the step that we use for "moving" towards the minimum of $E$. The learning rate can be thought of as the length of the arrows in figure 2.7. Usually $\eta \in \Re, 0 < \eta \leq 0.5$. Note that too large $\eta$ leads to oscillation around the minimum, while too small $\eta$ can lead to a slow convergence of the ANN.

Consider the second term in equation 2.8, namely $\frac{\partial E_d}{\partial w_{ji}}$. Note that $w_{ji}$ can influence the ANN only through $net_j$, i.e. the weighted sum of inputs for unit



Algorithm parameters: training_examples, $\eta, n_{in}, n_{out}, n_{hidden}$

Each training example is a pair of the form $\langle \vec{x}, \vec{t} \rangle$, where $\vec{x}$ is the vector of network input values, and $\vec{t}$ is the vector of target network output values.

$\eta$ is the learning rate (eg 0.05), $n_{in}$ is the number of network inputs, $n_{hidden}$ the number of units in the hidden layer, and $n_{out}$ the number of output units.

The input from unit i to unit j is denoted by $x_{ji}$, and the weight from unit i to unit j is denoted $w_{ji}$.

1. Create a feed-forward network with $n_{in}$ inputs, $n_{hidden}$ hidden units, and $n_{out}$ output units.

2. Initialize all network weights to small random numbers (eg between -0.05 and 0.05).

3. Until the termination condition is met, do

   - For each $\langle \vec{x}, \vec{t} \rangle$ in training_examples, do

     (a) *Propagate the input forward through the network:*

     Input the instance $\vec{x}$ to the network and compute the output $o_u$ of every unit $u$ in the network.

     *Propagate the errors backward through the network:*

     (b) For each network output unit $k$, calculate its error term $\delta_k$.

     $\delta_k \leftarrow transfer'(net_k)(t_k - o_k)$

     (c) For each hidden unit $h$, calculate its error term $\delta_h$

     $\delta_h \leftarrow transfer'(net_h) \sum_{k \in outputs} w_{kh} \delta_k$

     (d) Update each network weight $w_{ji}$,

     $w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$,

     where $\Delta w_{ji} = \eta \delta_j x_{ji}$.

Figure 2.8: Stochastic gradient descent backpropagation learning algorithm ([Mitchell, 1997, p. 98])

$j$. Therefore, we can use the chain rule to write ([Mitchell, 1997, p. 102])

$$\frac{\partial E_d}{\partial w_{ji}} = \frac{\partial E_d}{\partial net_j} \frac{\partial net_j}{\partial w_{ji}} \tag{2.9}$$

Provided that

$$\begin{aligned}
&\frac{\partial net_j}{\partial w_{ji}} = (net_j)dw_{ji} \\
&(net_j)dw_{ji} = \left( \sum_z w_{jz} x_{jz} \right) dw_{ji} \\
&(net_j)dw_{ji} = (w_{j0} \cdot x_{j0} + w_{j1} \cdot x_{j1} + \cdots + w_{ji} \cdot x_{ji} + \cdots + w_{jz} \cdot x_{jz})dw_{ji} \\
&(net_j)dw_{ji} = (0 \cdot x_{j0} + 0 \cdot x_{j1} + \cdots + 1 \cdot x_{ji} + \cdots + 0 \cdot x_{jz})dw_{ji} \\
&\Rightarrow \frac{\partial net_j}{\partial w_{ji}} = x_{ji}
\end{aligned} \tag{2.10}$$

the equation 2.9 reduces to

$$\frac{\partial E_d}{\partial w_{ji}} = \frac{\partial E_d}{\partial net_j} x_{ji} \tag{2.11}$$

The way $\delta_j = \frac{\partial E_d}{\partial net_j}$ is calculated differs between output nodes and the nodes that belong to the hidden layer. First, the output nodes will be treated and then we will discuss the hidden nodes case .

Just as $w_{ji}$ can influence the rest of the ANN only through $net_j$, $net_j$ can influence the ANN only through $o_j$. Therefore, we can invoke the chain rule again to write

$$\frac{\partial E_d}{\partial net_j} = \frac{\partial E_d}{\partial o_j} \frac{\partial o_j}{\partial net_j} \tag{2.12}$$

The first term in equation 2.12 $\frac{\partial E_d}{\partial o_j}$ can be rewritten as

$$\frac{\partial E_d}{\partial o_j} = -(t_j - o_j) \tag{2.13}$$

since

$$\frac{\partial E_d}{\partial o_j} = \left(\frac{1}{2}\sum_{z\in outputs}(t_z - o_z)^2\right) do_j$$
$$\left(\frac{1}{2}\sum_{k\in outputs}(t_z - o_z)^2\right) do_j = \frac{1}{2}((t_0 - o_0)^2 + (t_1 - o_1)^2 + \cdots + (t_j - o_j)^2 + \cdots + (t_z - o_z)^2)do_j$$
$$\left(\frac{1}{2}\sum_{k\in outputs}(t_z - o_z)^2\right) do_j = \frac{1}{2}(0 + 0 + \cdots + (t_j - o_j)^2 + \cdots + 0)do_j$$
$$\left(\frac{1}{2}\sum_{k\in outputs}(t_z - o_z)^2\right) do_j = \frac{1}{2}((t_j - o_j)^2)do_j \quad \left(\frac{1}{2}\sum_{k\in outputs}(t_z - o_z)^2\right) do_j = \frac{1}{2}(t_j^2 - 2t_j o_j + o_j^2)do_j$$
$$\left(\frac{1}{2}\sum_{k\in outputs}(t_z - o_z)^2\right) do_j = \frac{1}{2}(-2t_j + 2o_j) \quad \left(\frac{1}{2}\sum_{k\in outputs}(t_z - o_z)^2\right) do_j = (-t_j + o_j)$$
$$\left(\frac{1}{2}\sum_{k\in outputs}(t_z - o_z)^2\right) do_j = -(t_j - o_j) \tag{2.14}$$

The second term in equation 2.12, $\frac{\partial o_j}{\partial net_j}$, is resolved easily, if one remembers how nodes of an ANN process information by looking at figure 2.3. Due to the fact that $o_j = transfer(net_j)$, i.e. the weighted sum of inputs transferred by the transfer function, the second term becomes

$$\begin{aligned} \frac{\partial o_j}{\partial net_j} &= \frac{\partial transfer(net_j)}{\partial net_j} \\ \frac{\partial o_j}{\partial net_j} &= transfer'(net_j) \end{aligned} \tag{2.15}$$

Combining results of equations 2.8, 2.9, 2.10, 2.12, 2.13 and 2.15, the weight update $\Delta w$ for output nodes is expressed as

$$\begin{aligned} \Delta w &= -\eta\frac{\partial E_d}{\partial w_{ji}} \\ \Delta w &= -\eta\frac{\partial E_d}{\partial net_j}\frac{\partial net_j}{\partial w_{ji}} \\ \Delta w &= -\eta\left(\frac{\partial E_d}{\partial net_j}\right)\left(\frac{\partial net_j}{\partial w_{ji}}\right) \\ \Delta w &= -\eta\left(\left(\frac{\partial E_d}{\partial o_j}\right)\left(\frac{\partial o_j}{\partial net_j}\right)\right)(x_{ji}) \\ \Delta w &= -\eta\left((-(t_j - o_j))\left(transfer'(net_j)\right)\right)(x_{ji}) \\ \Delta w &= \eta \cdot (t_j - o_j) \cdot transfer'(net_j) \cdot x_{ji} \end{aligned} \tag{2.16}$$

Having explained the weight update rule for output nodes, we now will discuss how weights of hidden nodes are updated. As stated above (p. 48), of all sub-expressions of the weight update rule

$$\Delta w = -\eta \frac{\partial E_d}{\partial net_j} \frac{\partial net_j}{\partial w_{ji}} \tag{2.17}$$

only $\frac{\partial E_d}{\partial net_j}$ differs between output and hidden nodes. So, here we will describe only the derivation of this term, since all other expressions are same for both output and hidden nodes.

For hidden nodes, the derivation of $\frac{\partial E_d}{\partial net_j}$ must take into account the indirect ways in which $w_{ji}$ can influence the network outputs and hence $E_d$. For this reason, we will find it useful to refer to the set of all units immediately downstream of unit $j$ in the network (i.e. all units whose direct inputs include the output of unit $j$). We denote this set of units by $downstream(j)$. Notice that $net_j$ can influence the network outputs (and therefore $E_d$) only through the units in $downstream(j)$. Therefore we can write ([Mitchell, 1997, p. 103])

$$\begin{aligned}
\frac{\partial E_d}{\partial net_j} &= \sum_{k \in downstream(j)} \frac{\partial E_d}{\partial net_k} \frac{\partial net_k}{\partial net_j} \\
\frac{\partial E_d}{\partial net_j} &= \sum_{k \in downstream(j)} -\delta_k \frac{\partial net_k}{\partial net_j} \\
\frac{\partial E_d}{\partial net_j} &= \sum_{k \in downstream(j)} -\delta_k \frac{\partial net_k}{\partial o_j} \frac{\partial o_j}{\partial net_j} \\
\frac{\partial E_d}{\partial net_j} &= \sum_{k \in downstream(j)} -\delta_k w_{kj} \frac{\partial o_j}{\partial net_j} \\
\frac{\partial E_d}{\partial net_j} &= \sum_{k \in downstream(j)} -\delta_k w_{kj} transfer'(net_j)
\end{aligned} \tag{2.18}$$

Thus, the weight update rule for hidden nodes is equal to

$$\begin{aligned}
\Delta w &= -\eta \frac{\partial E_d}{\partial w_{ji}} \\
\Delta w &= -\eta \frac{\partial E_d}{\partial net_j} \frac{\partial net_j}{\partial w_{ji}} \\
\Delta w &= -\eta \left( \sum_{k \in downstream(j)} (-\delta_k w_{kj} transfer'(net_j)) \right) (x_{ji}) \\
\Delta w &= \eta \left( \sum_{k \in downstream(j)} (\delta_k w_{kj} transfer'(net_j)) \right) (x_{ji})
\end{aligned} \tag{2.19}$$

There are many improvements of this algorithm such as momentum term, weight decay etc described in the appropriate literature (eg Bishop [1996]). Nevertheless, MLP in combination with stochastic gradient descent learning algorithm is the most popular ANN technique used in practice. Another important feature of this learning algorithm is that it assumes a quadratic error function, hence it assumes there is only one minimum. In practice, the error function can have - apart from the global minimum - multiple local minima. There is a danger for the algorithm to land in one of the local minima and thus not be able to reduce the error to highest extent possible by reaching a global minimum.

### 2.1.7 Scaled conjugate gradient learning algorithm

Figure 2.9: Schematic illustration of fixed-step gradient descent for an error function which has substantially different curvatures along different directions ([Bishop, 1996, p. 265]).

Despite its popularity, the stochastic gradient descent backpropagation learning algorithm has several substantial drawbacks. Firstly, there is the need to specify the value of the learning rate $\eta$. Up to now, the optimal value is obtained empirically, without any theoretical knowledge about how the optimal $\eta$ is determined. The second, much more severe drawback of this method is slow convergence.

Figure 2.9 depicts the contours of $E$, for a hypothetical two-dimensional weight space, in which the curvature of $E$ varies significantly with direction. At most points on the error surface, the local gradient does not point directly towards the minimum. Gradient then takes many small steps to reach the minimum, and is clearly a very inefficient procedure ([Bishop, 1996, p. 264]).

The *scaled conjugate gradient* (SCG) learning algorithm does not have this dis-

advantage, but is far more complicated than the stochastic gradient descent backpropagation algorithm. The explanation in this section will firstly introduce the *line search*, a concept upon which the further explained *conjugate gradient* (CG) learning algorithm is based and concludes with explanation of the SCG algorithm, that is an extension of the CG algorithm.

CG and SCG algorithms involve taking a sequence of steps through weight space. It is convenient to consider each of these steps in two parts. First we must decide the direction in which to move, and second, we must decide how far to move in that direction. With simple gradient descent, the direction of each step is given by the local negative gradient of the error function, and the step size is determined by an arbitrary learning rate parameter. We might expect that a better procedure would be to move along the direction of the negative gradient to find the point at which the error is minimized. More generally we can consider some *search direction* in weight space, and then find the minimum of the error function along that direction. This procedure is referred to as a *line search*, and it forms the basis for several algorithms which are considerably more powerful than gradient descent. We first consider how line searches can be implemented in practice ([Bishop, 1996, p. 272]).

Suppose that at step $t$ in some algorithm the current weight vector is $w^t$, and we wish to consider a particular search direction $d^t$ through weight space. The minimum along the search direction then gives the next value for the weight vector:

$$w^{t+1} = w^t + \lambda^t d^t \tag{2.20}$$

where the parameter $\lambda^t$ is chosen to minimize

$$E(\lambda) = E\left(w^t + \lambda d^t\right) \tag{2.21}$$

This gives us an automatic procedure for setting the step length, once we have chosen the search direction ([Bishop, 1996, p. 272]). The line search represents a one-dimensional minimization problem. A simple approach would be to proceed along the search direction in small steps, evaluating the error function at each new position, and stop when the error starts to increase (Hush and Salas [1988]).

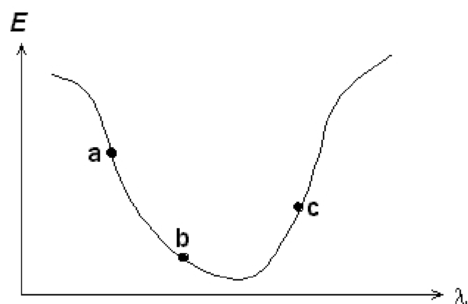



Figure 2.10: Bracketing the minimum ([Bishop, 1996, p. 273]).

It is possible, however, to find very much more efficient approaches (Press et al. [1992]). Consider first the issue of whether to make use of gradient information in performing a line search. We have already argued that there is generally a substantial advantage to be gained from using gradient information for the general problem of seeking the minimum of the error function $E$ in the $W$-dimensional weight space. For the sub-problem of line search, however, the argument is somewhat different. Since this is now a one-dimensional problem, both the value of the error function and the gradient of the error function each represent just one piece of information. An error function calculation requires one forward propagation and hence needs $\sim 2NW$ operations, where $N$ is the number of patterns in the data set. An error function gradient evaluation, however, requires a forward propagation, a backward propagation, and a set of multiplications to form the derivatives. It therefore needs $\sim 5NW$ operations, although it does allow the error function itself to be evaluated as well. On balance the line search is slightly more efficient if it makes use of error function evaluations only ([Bishop, 1996, pp. 272–273]). Each line search proceeds in two stages. The first stage is to *bracket* the minimum by finding three points $a < b < c$ along the search direction such that $E(a) > E(b)$ and $E(c) > E(b)$ as shown in figure 2.10. Since the error function is continuous, this ensures that there is a minimum somewhere in the interval $(a, c)$ (Press et al. [1992]). The second stage is to locate the minimum itself. Since the error function is smooth

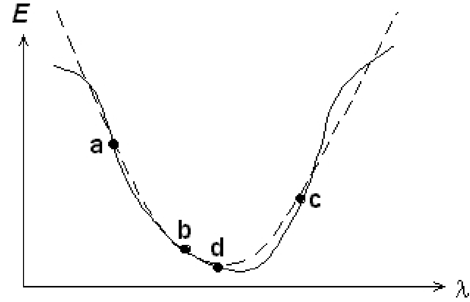



Figure 2.11: Parabolic interpolation used to perform line-search minimization ([Bishop, 1996, p. 274]).

and continuous, this can be achieved by a process of parabolic interpolation. This involves fitting a quadratic polynomial to the error function evaluated at three successive points, and then moving to the minimum of the parabola, as illustrated in figure 2.11. The process can be repeated by evaluating the error function at the new point, and then fitting a new parabola to this point and two of the previous points ([Bishop, 1996, pp. 272–273]).

To apply line search to the problem of error function minimization, we need to choose a suitable search direction at each stage of the algorithm. Suppose we have already minimized along a search direction given by the local negative gradient vector. We might suppose that the search direction at the next iteration will be given by the negative gradient vector at the new position. However, the use of successive gradient vectors turns out in general not to represent the best choice of search direction. To see why, we note that at the minimum of the line search we have (see equation 2.21)

$$\frac{\partial}{\partial \lambda} E(w^t + \lambda d^t) = 0 \tag{2.22}$$

which gives

$$\left(\nabla E^{t+1}\right)^T d^t = 0 \tag{2.23}$$

Thus, the gradient at the new minimum is orthogonal to the previous search direction, as illustrated geometrically in figure 2.12. Choosing successive search directions to be the local negative gradient directions can lead to the problem

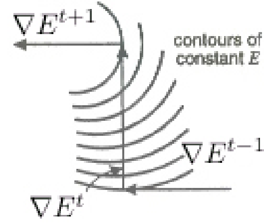



Figure 2.12: Orthogonality of new gradient after line minimization ([Bishop, 1996, p. 275]).

already indicated in figure 2.9 in which the search point oscillates on successive steps while making little progress towards the minimum. The algorithm can then take many steps to converge, even for a quadratic error function ([Bishop, 1996, p. 275]). The solution to this problem lies in choosing the successive

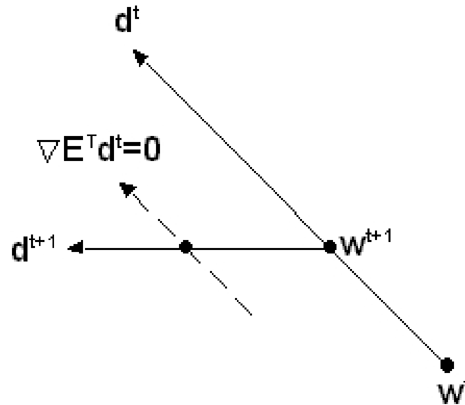



Figure 2.13: The concept of conjugate directions ([Bishop, 1996, p. 276]).

search directions $d^t$ such that, at each step of the algorithm, the component of the gradient parallel to the previous search direction, which has just been made zero, is unaltered. This is illustrated in figure 2.13. Suppose we have already performed a line minimization along the direction $d^t$, starting from the point $w^t$, to give the new point $w^{t+1}$. Then, at the point $w^{t+1}$ we have

$$\nabla E(w^{t+1})^T d^t = 0 \tag{2.24}$$

We now choose the next search direction $d^{t+1}$ such that, along this new direc-

tion, we retain the property that the component of the gradient parallel to the previous search direction remains zero. Thus we require that

$$\nabla E(w^{t+1} + \lambda d^{t+1})^T d^t = 0 \tag{2.25}$$

as shown in figure 2.13. If we now expand 2.25 to first order in $\lambda$, and note that the zeroth-order term vanishes as a consequence of equation 2.24, we obtain

$$(d^{t+1})^T H d^t = 0 \tag{2.26}$$

where H is the Hessian matrix evaluated at the point $w^{t+1}$. If the error surface is quadratic, this relation holds for arbitrary values of $\lambda$ in equation 2.25, since the Hessian matrix is constant, and higher-order terms in the expansion of 2.25 in powers of $\lambda$ vanish. Search directions which satisfy 2.26 are said to be *non-interfering* or *conjugate* ([Bishop, 1996, pp. 275–276]).

The conjugate gradient algorithm trains the ANN in such a way, that at a certain iteration of the algorithm, the chosen search direction is conjugate to all previous search directions. The conjugate gradient algorithm is descried in detail by Bishop ([Bishop, 1996, pp. 276–282]).

The *scaled conjugate gradient* (SCG) algorithm developed by Møller (Møller [1993]) is an extension the conjugate gradient algorithm outlined above. Its main advantages is the avoidance of the line-search procedure (that is present in the CG algorithm) so that SCG is faster than both CG and stochastic gradient descent backpropagation algorithms. Before proceeding to the explanation of Møller's SCG algorithm [2], some related notation is given. Let $N$ be the number of weights in a neural network. A $N \times N$ matrix $A$ is said to be *positive definite*, if

$$y^T A y > 0, \quad \forall y \in \mathfrak{R}^N \tag{2.27}$$

Let $p_1, p_2, \ldots, p_k$ be a set of non-zero weight vectors in $\mathfrak{R}$. The set is said to be a *conjugate system* with respect to a non-singular symmetric $N \times N$ matrix $A$ if

$$p_i^T A p_j = 0, \quad i \neq j, i = 1, 2, \ldots, k \tag{2.28}$$

[2] The explanation of the SCG algorithm is based upon [Møller, 1993, pp. 61–74].

holds. The set of points $w$ in $\mathfrak{R}$ satisfying

$$w = w_1 + \alpha_1 p_1 + \cdots + \alpha_k p_k, \quad \alpha_i \in \mathfrak{R} \tag{2.29}$$

where $w_1$ is a point in weight space and $p_1, p_2, \ldots, p_k$ is a subset of a conjugate system, is called *k-plane* or $\pi_k$ (Møller [1993]). The SCG algorithm is shown in figure 2.14. The SCG algorithm works in principle according to following scheme: at iteration $k$ of the algorithm, choose a search direction $p_k$ and step size $\alpha_k$, such that $E(w_k + \alpha_k p_k) < E(w_k)$, i.e. the movement into the direction $p_k$ by $\alpha_k$ corresponds to the movement to the minimum of the error function. The process is to be repeated until $\nabla E(w_k) = 0$.
The SCG algorithm follows this strategy but uses information from the second-order approximation of the error function $E$ given by

$$E(w + y) \approx E(w) + E'(w)^T y + \frac{1}{2} y^T E''(w) y \tag{2.30}$$

The quadratic approximation to $E$ in a neighbourhood of a point w is

$$E_{qw}(y) = E(w) + E'(w)^T y + \frac{1}{2} y^T E''(w) y \tag{2.31}$$

In order to determine minima to $E_{qw}(y)$ the critical points for $E_{qw}(y)$ must be found, i.e. the points where

$$E'_{qw}(y) = E''(w) y + E'(w) = 0 \tag{2.32}$$

The critical points are the solution to the linear system defined by the equation 2.32. According to Hestenes and Stiefel [1952] and Johansson et al. [1991], the solution can be simplified considerable, if a conjugate system is available. Assume $p_1, p_2, \ldots, p_N$ is a conjugate system, where $N$ is the number of weights of an ANN. The step from a starting point $y_1$ to a critical point $y_*$ can be expressed as a linear combination of $p_1, p_2, \ldots, p_N$:

$$y_* - y_1 = \sum_{i=1}^{N} \alpha_i p_i, \quad \alpha_i \in \mathfrak{R} \tag{2.33}$$

Multiplying equation 2.33 with $p_j^T E''(w)$ and substituting $E'(w)$ for $-E''(w) y_*$

1. Choose weight vector $w_1$ and scalars $0 < \sigma \leq 10^{-4}$, $0 < \lambda_1 \leq 10^{-6}$ and $\bar{\lambda}_1 = 0$. Set $p_1 = r_1 = -\nabla E(w_1), k = 1, success = true$.

2. If $success = true$ then calculate second order information:

$$\sigma_k = \frac{\sigma}{|p_k|} \qquad s_k = \frac{\nabla E(w_k + \sigma_k p_k) - \nabla E(w_k)}{\sigma_k} \qquad \delta_k = p_k^T s_k.$$

3. Scale $s_k$: $\delta_k = \delta_k + (\lambda_k - \bar{\lambda}_k)|p_k|^2$.

4. If $\delta_k \leq 0$, then make the Hessian matrix positive definite:

$$\bar{\lambda}_k = 2(\lambda_k - \frac{\delta_k}{|p_k|^2}) \qquad \delta_k = -\delta_k + \lambda_k |p_k|^2 \quad \lambda_k = \bar{\lambda}_k$$

5. Calculate step size: $\mu = p_k^T r_k \quad \alpha = \frac{\mu_k}{\delta_k}$.

6. Calculate the comparison parameter: $\Delta_k = \frac{2\delta_k [E(w_k) - E(w_k + \alpha_k p_k)]}{\mu_k^2}$

7. If $\Delta_k \geq 0$

   - then a successful reduction in error can be made:

     $$w_{k+1} = w_k + \alpha_k p_k \quad r_{k+1} = -\nabla E(w_{k+1}) \quad \bar{\lambda}_k = 0, success = true$$

     If $kmodN = 0$

     - then restart algorithm: $p_{k+1} = r_{k+1}$
     - else create new conjugate direction:

       $$\beta_k = \frac{|r_{k+1}|^2 - r_{k+1}^T r_k}{|r_k|^2} \quad p_{k+1} = r_{k+1} + \beta_k p_k$$

     If $\Delta_k \geq 0.75$ then reduce the scale parameter: $\lambda_k = \frac{1}{4}\lambda_k$.

   - else ($\Delta_k < 0$), a reduction in error is not possible: $\bar{\lambda}_k = \lambda_k, success = false$

8. If $\Delta_k < 0.25$ then increase the scale parameter: $\lambda_k = \lambda_k + \frac{\delta_k(1-\Delta_k)}{|p_k|^2}$.

9. If the steepest descent direction $r_k \neq 0$ then set $k = k + 1$ and go to step 2,

   else terminate and return $w_{k+1}$ as the desired minimum.

Figure 2.14: Scaled conjugate gradient learning algorithm (Møller [1993])

gives

$$p_j^T(-E'(w) - E''(w)y_1) = \alpha_j p_j^T E''(w) p_j \Rightarrow \tag{2.34}$$

$$\alpha_j = \frac{p_j^T(-E'(w) - E''(w)y_1)}{p_j^T E''(w) p_j} = \frac{-p_j^T E'_{qw}(y_1)}{p_j^T E''(w) p_j} \tag{2.35}$$

The critical point $y_*$ can be determined in $N$ iterative steps using equations 2.33 and 2.35. Unfortunately $y_*$ is not necessarily a minimum, but can be a saddle point or a maximum. Only if the Hessian matrix $E''(w)$ is positive definite then $E_{qw}(y)$ has a unique global minimum. This can be realized by (derivation is given in Møller [1993])

$$E_{qw}(y) = E_{qw}(y_*) + \frac{1}{2}(y - y_*)^T E''(w)(y - y_*) \tag{2.36}$$

If follows from 2.36 that if $y_*$ is a minimum then $\frac{1}{2}(y - y_*)^T E''(w)(y - y_*) > 0$ for every $y$, hence $E''(w)$ has to be positive definite.

The intermediate points $y_{k+1} = y_k + \alpha_k p_k$ given by the iterative determination of $y_*$ are in fact minima for $E_{qw}(y)$ restricted to every k-plane $\pi_k : y = y_1 + \alpha_1 p_1 + \cdots + \alpha_k p_k$ (Hestenes and Stiefel [1952]). These points can be determined recursively using a theorem 1.

**Theorem 1** *Let $p_1, \ldots, p_N$ be a conjugate system and $y_1$ a point in weight space. Let the points $y_2, \ldots, y_{N+1}$ be recursively defined by*

$$y_{k+1} = y_k + \alpha_k p_k \tag{2.37}$$

*with $\alpha_k = \frac{\mu_k}{\delta_k}$, $\mu_k = -p_k^T(y_k)$ and $\delta_k = p_k^T E''(w) p_k$. Then $y_{k+1}$ minimizes $E_{qw}$ restricted to the k-plane $\pi_k$ given by $y_1$ and $p_1, \ldots, p_k$ (Hestenes and Stiefel [1952]).*

The values $p_k$ in 1 can also be obtained using the theorem 2.

**Theorem 2** *Let $y_1$ be a point in weight space and $p_1$ and $r_1$ equal to the steepest descent vector $-E'_{qw}(y_1)$. Define $p_{k+1}$ recursively by*

$$p_{k+1} = r_{k+1} + \beta_k p_k, \tag{2.38}$$

*where $r_{k+1} = -E'_{qw}(y_{k+1})$, $\beta_k = \frac{|r_{k+1}|^2 - r_{k+1}^T r_k}{r_k^T}$ and $y_{k+1}$ is the point generated in theorem 1. Then $p_{k+1}$ is the steepest descent vector to $E_{qw}$ restricted to the*

*$(N-k)$-plane $\pi_{N-k}$ conjugate to $\pi_k$ given by $y_1$ and $p_1, \ldots, p_k$ (Hestenes and Stiefel [1952]).*

The functioning of the "classical" CG algorithm (the one briefly presented above) is based upon theorems 1 and 2 (Hestenes and Stiefel [1952]).
The main difference between CG and SCG algorithm is the way, how second-order derivative $E''(w)$ is obtained in the term $p_k^T E''(w) p_k$ in theorem 1. In SCG algorithm, the computational costly evaluation of this derivative is avoided by using a non-symmetric approximation of $E''(w_k)p_k$ of the form

$$s_k = E''(w_k)p_k \approx \frac{E'(w_k + \sigma_k p_k) - E'(w_k)}{\sigma_k}, \quad 0 < \sigma_k \ll 1. \tag{2.39}$$

The approximation tends in the limit to the true value of $E''(w_k)p_k$ (Møller [1993]). The calculation complexity and memory usage of $s_k$ are respectively $\mathcal{O}(PN)$ [3] and $\mathcal{O}(N)$ (instead of $\mathcal{O}(N^2)$ and $\mathcal{O}(PN^2)$. The direct incorporation of this formula into the classical CG algorithm yields poor results for two reasons:

- The algorithm works only for functions with positive definite Hessian matrices (this is not always the case) and
- The quadratic approximation $E_{qw}$, on which the algorithm works, can be very poor when the current point is far from the desired minimum (Gill et al. [1981]).

Thus, some additional mechanisms must be provided in order for the SCG algorithm to work. Møller (Møller [1993]) proposes the use of *model-trust approach*, i.e. to add a term to the definition of $s_k$ is equation 2.39 in order to regulate the indefiniteness of $E''(w_k)$:

$$s_k = \frac{E'(w_k + \sigma_k p_k) - E'(w_k)}{\sigma_k} + \lambda_k p_k \tag{2.40}$$

$\lambda_k$ is adjusted in each iteration looking at the sign of $\delta_k$, which directly reveals if $E''(w_k)$ is not positive definite. If $\delta_k \leq 0$ then the Hessian is not positive

[3] For the definition of the $\mathcal{O}$-notation refer to appendix H

definite and $\lambda_k$ is raised and $s_k$ is estimated again. If the new $s_k$ is renamed as $\bar{s}_k$ and the raised $\lambda_k$ as $\bar{\lambda}_k$ then $\bar{s}_k$ is

$$\bar{s}_k = s_k + (\bar{\lambda}_k - \lambda_k)p_k. \tag{2.41}$$

Assume in a given iteration that $\delta_k \leq 0$. It is possible to determine how much $\lambda_k$ should be raised in order to get $\delta_k > 0$. If the new $\delta_k$ is renamed as $\bar{\delta}_k$ then

$$\begin{aligned} &\bar{\delta}_k = p_k^T \bar{s}_k = p_k^T(s_k + (\bar{\lambda}_k - \lambda_k)p_k) = \delta_k + (\bar{\lambda}_k - \lambda_k)|p_k|^2 > 0 \Rightarrow \\ &\bar{\lambda} > \lambda_k - \frac{\delta_k}{|p_k|^2}. \end{aligned} \tag{2.42}$$

2.42 implies that if $\lambda_k$ is raised with more than $-\frac{\delta_k}{|p_k|^2}$ then $\bar{\delta}_k > 0$. The question is: how much should $\bar{\lambda}_k$ be raised to get an optimal solution? This question can not yet be answered, but it is clear that $\bar{\lambda}_k$ in some way should depend on $\lambda_k, \delta_k$ and $|p_k|^2$. A choice found to be reasonable is

$$\bar{\lambda}_k = 2\left(\lambda_k - \frac{\delta_k}{|p_k|^2}\right) \tag{2.43}$$

This leads to

$$\begin{aligned} \bar{\delta}_k =& \delta_k + (\bar{\lambda}_k - \lambda_k)|p_k|^2 = \delta_k + (2\lambda_k - 2\frac{\delta_k}{|p_k|^2} - \lambda_k)|p_k|^2 \\ =& -\delta_k + \lambda_k|p_k|^2 > 0. \end{aligned} \tag{2.44}$$

The step size is given by

$$\alpha_k = \frac{\mu_k}{\delta_k} = \frac{\mu_k}{p_k^T s_k + \lambda_k|p_k|^2}, \tag{2.45}$$

with $s_k$ given by equation 2.39. The values of $\lambda_k$ directly scale the step size in such a way that the bigger $\lambda_k$ is the smaller the step size, which agrees well with out intuition of the function of $\lambda_k$. The quadratic approximation $E_{qw}$, on which the algorithm works, may not always be a good approximation to $E(w)$ since $\lambda_k$ scales the Hessian matrix in an artificial way. A mechanism to raise and lower $\lambda_k$ is needed which gives a good approximation, even when the Hessian is positive definite. Define

$$\Delta_k = \frac{E(w_k) - E(w_k + \alpha_k p_k)}{E(w_k) - E_{qw}(\alpha_k p_k)} = \frac{2\delta_k(E(w_k) - E(w_k + \alpha_k p_k))}{\mu_k^2}. \tag{2.46}$$

Here $\Delta_k$ is a measure of how well $E_{qw}(\alpha_k p_k)$ approximates $E(w_k + \alpha_k p_k)$ in the sense, that the closer $\Delta_k$ is to 1, the better is the approximation. $\lambda_k$ is raised and lowered following the formula

$$\begin{cases} \text{if } \Delta_k > 0.75 \text{ then} & \lambda_k = \frac{1}{4}\lambda_k \\ \text{if } \Delta_k < 0.25 \text{ then} & \lambda_k = \lambda_k + \frac{\delta_k(1-\Delta_k)}{|p_k|^2} \end{cases}. \tag{2.47}$$

The formula for $\Delta_k < 0.25$ increases $\lambda_k$ such that the new step size is equal to the minimum of a quadratic polynomial fitted to $E'(w_k)^T p_k$, $E(w_k)$ and $E(w_k + \alpha_k p_k)$ (Williams [1991]).

## 2.2 Pre- and postprocessing

### 2.2.1 Curse of dimensionality

One of the most important steps in designing a neural network is the choice of appropriate data pre- and postprocessing. This is necessary due to a phenomenon called *curse of dimensionality.* Consider a neural network that should model a relationship between input values $x_1, \ldots, x_d$ and the output variable $y$. One way to represent the training data would be to define intervals for input variables and then to classify the data records by saying, in which interval the values $x_1, \ldots, x_d$ of a particular record lie. For the case of $d = 3$, this results in the division of input space into a high number of small boxes as shown in figure 2.15. Each box corresponds to an $d$-dimensional "interval". If the number of divisions (interval) per one dimension is $M$, then the total number of boxes equals to $M^d$, thus growing exponentially with the number of dimensions. Since each point has to contain at least one data point, the number of training data also grows exponentially when using this approach ([Bishop, 1996, pp. 7–8]). Thus, contrary to the intuitive assumption that additional data should improve the performance of the neural network, it is not necessary the case. In fact, often the reduction of dimensionality of the training data is necessary for a proper functioning of the neural network. This will be illustrated using another example. Consider a neural network that should recognize some shape in a monochrome

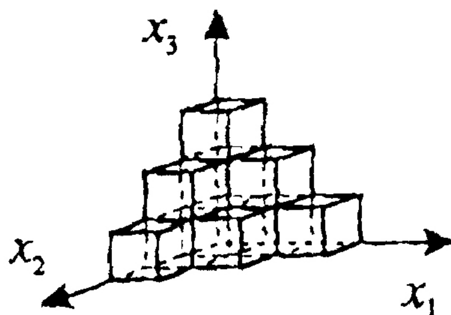



Figure 2.15: One way to specify a mapping from a $d$-dimensional space $x_1, \ldots, x_d$ to an output variable $y$ is to divide the input space into a number of cells, as indicated here for the case $d = 3$, and to specify the value of $y$ for each of the cells. The major problem with this approach is that the number of cells and hence the number of example data points required, grows exponentially with $d$, a phenomenon known as the curse of dimensionality ([Bishop, 1996, p. 8]).

picture with a resolution of $256 \times 256$ pixels. The most straightforward approach would be to take each pixel as an input value to a neural network[4]. But in this case the number of weights in the hidden layer would be enormous: if there is one input node for each pixel, there are $256 \times 256 = 65536$ input nodes, each of them being connected to each node of the hidden layer. Hence, each node of the hidden layer would have 65536 connections (therefore 65536 weights) to the input layer. This approach would require great amounts of training data and huge computational resources, while it is highly questionable that - when trained - such a neural network would perform well ([Bishop, 1996, p. 297]).

In this case, it makes more sense to preprocess the picture in order to extract the features crucial for the recognition of the shape and then to apply[5] neural network on these features. So, it is known that so-called *Fourier descriptors*

[4]"Neural network" means here a neural network of any design except shared weights architecture described on p. 77.

[5]It is assumed that basic preprocessing (segmentation, noise reduction, edge detection etc) of the image data is already done.

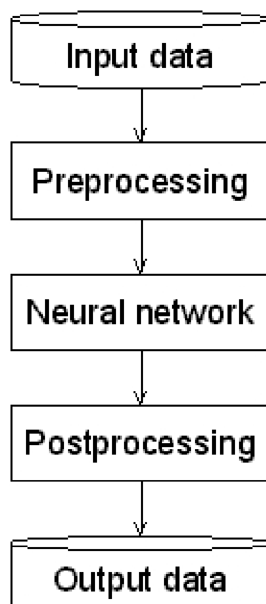



Figure 2.16: Schematic illustration of the use of pre-processing and post-processing in conjunction with a neural network mapping ([Bishop, 1996, p. 296]).

capture the contour properties of a figure in 10 - 20 (depending on complexity of the shape) real numbers and are even invariant to rotation of the figure. The neural network could be trained to recognize the figure using these Fourier descriptors[6].

### 2.2.2 Operation of a neural network

The general scheme for using a neural network is shown in figure 2.16. Continuing the above example of a shape recognition neural network, the input data would correspond to the values of 65536 pixels. The preprocessing mechanism would calculate the Fourier descriptors. These would be fed to the neural network, and the neural network would produce some output values. Often, the neural networks are designed so that each possible output value corresponds to one output node (e.g. if the recognized shape is a rectangle, then output node

[6] An explanation of Fourier descriptors can be found in [Seul et al., 2000, p. 152].

1 is equal to 1 while all others are zero; if the recognized shape is a circle, then output node 2 is equal to 1 while all others are zero etc). The postprocessing mechanism would transform such encodings into a more comprehensible form (e.g. by displaying a rectangle, circle or printing some verbal output).

### 2.2.3 Methods of pre- and postprocessing

The methods of data pre- and postprocessing include[7]

**Input normalization and encoding** This technique is used to rescale the input variables. This is often useful, if different variables have typical values which differ significantly. In a system monitoring a chemical plant, two of the inputs might represent a temperature and a pressure respectively. Depending on the units in which each of these is expressed, they may have values which differ by several orders of magnitude. Furthermore, the typical sizes of the inputs may not reflect their relative importance in determining the required outputs ([Bishop, 1996, p. 298]).

The following transformation would transform the original input variables $x_1, \ldots, x_d$ into normalized variables $\tilde{x}_1, \ldots, \tilde{x}_d$ that have values lying in similar ranges ($N$ denotes the number of patterns in training set and $x_i^n$ the value of variable $x_i$ of the $n$-th pattern):

$$
\begin{aligned}
& i = 1 \ldots d \\
& \bar{x}_i = \frac{1}{N} \sum_{n=1}^{N} x_i^n \quad \sigma_i^2 = \frac{1}{N-1} \sum_{n=1}^{N} (x_i^n - \bar{x}_i)^2 \\
& \tilde{x}_i^n = \frac{x_i^n - \bar{x}_i}{\sigma_i}
\end{aligned}
$$

The transformed values ($\tilde{x}_i^n$) will have zero mean and unit standard deviation over the transformed training set.

**Treatment of discrete data** Continuous (real) variables can be fed directly into a neural network without preprocessing. This applies also to ordinal discrete variables (those values, which have a natural ordering, e.g. age).

[7] This section (including subsections) is based upon [Bishop, 1996, pp. 295–331]

Categorical discrete variables (e.g. color "red", "green", "blue") are usually encoded using the *1-of-c* coding ([Bishop, 1996, p. 300]). That means that there is exactly one node corresponding to a particular value of the variable (e.g. the colour variable may be represented by three input nodes $x = (x_1, x_2, x_3)$, which are equal to $(1, 0, 0)$ if the colour is red, $(0, 1, 0)$ and $(0, 0, 1)$ if the colour is green or blue respectively).

**Treatment of missing data** Often the patterns of the training set are incomplete, i.e. there are records in the training set, which have "empty" values. If the amount of corrupted records is too high to ignore them, missing values are usually estimated using multiple regression analysis (i.e. to get the missing values by running a regression over the other variables using the available data, [Bishop, 1996, p. 301]).

**Feature selection** This method reduces the dimensionality of data by selecting a subset of available variables and discarding the remaining ones (see section 2.2.4).

**Principal component analysis** Principal component analysis aims at reduction of dimensionality while preserving as much information contained in the data as possible by combining together several input variables (see section 2.2.5).

**Invariances and prior knowledge** Sometimes it is known that the output variable should remain unchanged, when the input is subject to various transformations. This may require additional provisions. On the other side, knowledge of the application domain of the neural network can be of great benefit for the design of it (see section 2.2.6).

### 2.2.4 Feature selection

Feature selection servers the purpose of reduction of the dimensionality of the input data by discarding those input variables, which carry little information or carry only the information already contained in other input variables. The selection of relevant input variables (features) must, firstly, define how much

information a particular feature subset does carry and secondly, determine the optimal feature subset with respect to the information it carries.

### Selection criteria

The selection criterion should measure the "goodness" of a particular subset of features. Ideally, this would involve training the neural network on that feature subset, running the trained network against a test set and measuring the performance of the network. This is not possible in most cases, for neural network training often requires significant amount of time and due to the high number of different feature subsets. Therefore, simpler methods are used in practice for this task, in order to be able to explore a large number of feature combinations. For regression problems, a single-layer-network with linear output units is trained on a certain feature subset and its error value (sum of squared errors) is taken as a measure of goodness of that feature subset. Contrary to MLPs, single-layer-networks can be trained relatively quickly.

For classification problems, the class separability is used as a measure of goodness of a feature subset. A widely used measure of class separation is the Fisher's criterion function (see equation I.36 on p. 187 and explanation in appendix I). Despite the curse of dimensionality, it cannot be expected that the performance of the neural network increases as features are deleted. Performance measures $J$ with respect to feature subset $X$ and a larger feature subset $X^+$ usually satisfy the monotonicity criterion

$$J(X^+) \geq J(X) \tag{2.48}$$

hence, deletion of features cannot reduce the error rate. Therefore, such criteria (e.g. the Mahalanobis distance) can only be used to compare different feature subset, but not to determine the optimal one. Instead, search techniques presented in the following section must be used to obtain the optimal feature subset.

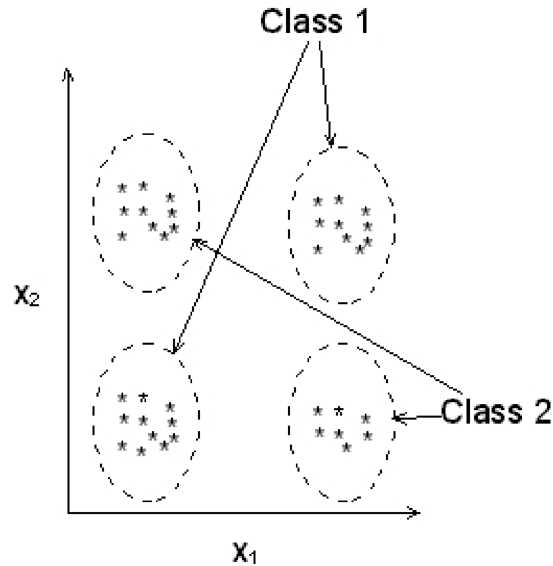



Figure 2.17: Example of data from two classes as described by two feature variables $x_1$ and $x_2$. If the data was described by either feature alone then there would be strong overlap of the two classes, while if both features are used, as shown here, then the classes are well separated ([Bishop, 1996, p. 307])

**Search techniques**

The need for the application of search technique for finding the optimal feature subset arises from the fact that there are many possible feature subsets: for a total number $d$ of features it equals to $2^d$ (2 because each feature may be present or absent in the subset). Thus, for all but very small input data dimensions, exhaustive search (i.e. trying out all possible combinations of features) is computationally impossible. Exhaustive search is applied only then, if there is no alternative to it, such as the example in figure 2.17. Either feature alone does contributes little information needed for discriminating the classes, but both of them allow to distinguish the classes easily. Such cases can occur with an arbitrary number of input variables (in this case 2).

If the selection criterion satisfies equation 2.48, then the so-called *branch and bound technique* is guaranteed to find an optimal feature subset without per-

forming an exhaustive search, since many potential subsets can be ruled out without the need to evaluate them (a thorough treatment of branch and bound technique can be found in Narendra and Fukunaga [1977]).

If this technique is still too computationally expensive, *sequential search techniques* can be applied. In this case, the algorithm begins by considering each individual input variable and choosing the one with the highest value of the selection criterion. At each stage of the algorithm, the feature yielding the largest selection criterion value is added to the feature subset (see figure 2.18). A variation of this algorithm called *sequential backward elimination* starts with

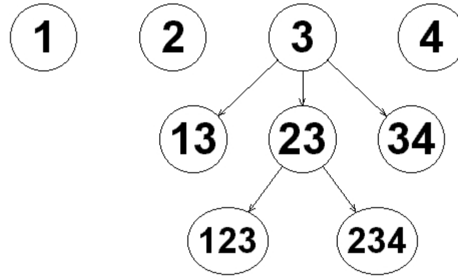



Figure 2.18: Sequential forward selection illustrated for a set of four features, denoted by 1,2,3 and 4. The single best feature variable is chosen first, and then features are added one at a time such that at each stage the variable chosen is the one which produces the greatest increase in the criterion function ([Bishop, 1996, p. 309]).

a "subset" containing all features and removes one feature (that whose removal results in least reduction of the selection criterion value) per step. The advantage of sequential backward elimination is the fact that it can cope with situations like one shown in figure 2.17), where individual features contribute little to increasing the selection criterion value if taken alone, but improve the performance of neural network greatly when taken together. There are many possible combinations of both algorithms.

### 2.2.5 Principal component analysis

The principal component analysis aims at dimensionality reduction of input data. One method was already mentioned, namely the Fisher's discriminant analysis described in appendix I. This method has a drawback, namely that it reduces dimensionality from $d$ (number of dimensions of the input data) to exactly $(C-1)$ ($C$ being the number of classes). Methods of principal component analysis allow dimensionality reduction to an arbitrary number of features. They do not take into account target data information and rely solely upon input data. Therefore, these methods can produce suboptimal results and there is no guarantee, that the dimensionality reduction will not "reduce the important information away".

#### Karhunen-Loeve transformation

The goal of principal component analysis is to map $d$-dimensional vectors $x_i$ onto m-dimensional vectors $z_i$ with $m < d$. The vector $x$ can be represented as a linear combination of a set of $d$ orthonormal vectors $u_i$

$$x = \sum_{i=1}^{d} z_i u_i \tag{2.49}$$

where the vectors $u_i$ satisfy the orthonormality relation

$$u_i^T u_j = \delta_{ij} \tag{2.50}$$

in which $\delta_{ij}$ is the Kronecker delta symbol. Explicit expressions for the coefficients $z_i$ can be found by using the equation 2.50:

$$z_i = u_i^T x \tag{2.51}$$

The dimensionality reduction happens in the following way: only $m$ ($m < d$) coefficients $z_i$ are used, while the remaining coefficients are replaced by constants $b_i$ so that each vector $x$ is approximated by an expression of the form

$$\tilde{x} = \sum_{i=1}^{m} z_i u_i + \sum_{i=m+1}^{d} b_i u_i \tag{2.52}$$

Then the algorithm aims at choosing the basis vectors $u_i$ and the coefficients $b_i$ such that the approximation given by 2.52 with the values $z_i$ determined using equation 2.51 gives the best approximation to the original vector $x$ on average for the whole data set. The error in the vector $x$ introduced by the dimensionality reduction is given by

$$x_i - \tilde{x}_i = \sum_{j=m+1}^{d} (z_i, j - b_j) u_j \tag{2.53}$$

The best approximation is one that minimizes the sum of the squares of the errors over the whole data set. Thus, the value to be minimized is equal to

$$E_M = \frac{1}{2} \sum_{n=1}^{N} ||x_n - \tilde{x}_n||^2 = \frac{1}{2} \sum_{n=1}^{N} \sum_{i=m+1}^{d} (z_{n,j} - b_i)^2 \tag{2.54}$$

where $N$ denotes the number of samples in the data set. If the derivative of $E_M$ with respect to $b_i$ is set to zero, then

$$b_i = \frac{1}{N} \sum_{n=1}^{N} z_i^n = u_i^T \bar{x} \tag{2.55}$$

$$\bar{x} = \frac{1}{N} \sum_{n=1}^{N} x_i \tag{2.56}$$

Using the equations 2.51 and 2.55 the sum-of-squares error can be rewritten as

$$E_M = \frac{1}{2} \sum_{i=m+1}^{d} \sum_{n=1}^{N} \left( u_i^T (x_n - \bar{x}) \right)^2 \tag{2.57}$$

$$= \frac{1}{2} \sum_{i=m+1}^{d} u_i^T \Sigma u_i \tag{2.58}$$

where $\Sigma$ is the covariance matrix of the set of vectors $x_i$ and is given by

$$\Sigma = \sum_{n} (x_n - \bar{x})(x_n - \bar{x})^T \tag{2.59}$$

There now remains the task of minimizing $E_M$ with respect to the choice of basis vectors $u_i$. As shown in [Bishop, 1996, pp. 454–456], the minimum occurs when the basis vectors satisfy

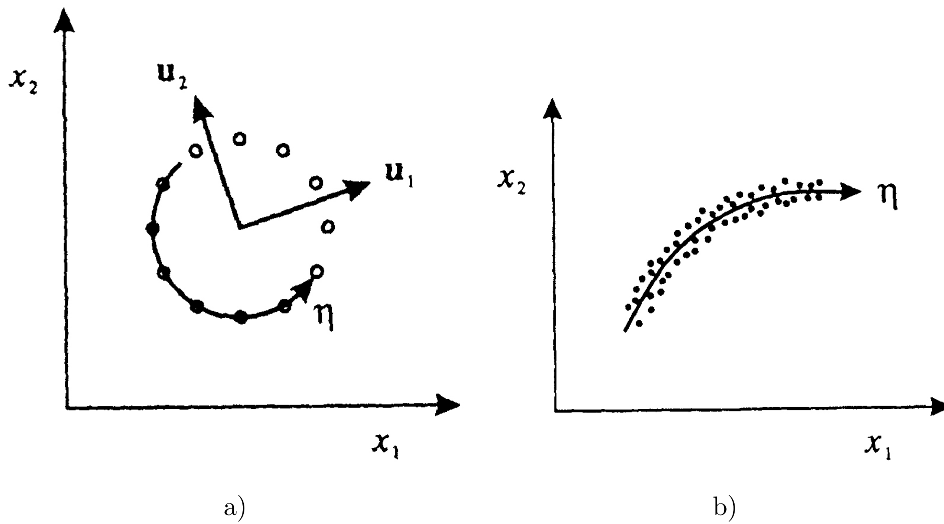
$$\Sigma u_i = \lambda_i u_i \tag{2.60}$$

so that they are the eigenvectors of the covariance matrix. Note that, since the covariance matrix is real and symmetric, its eigenvectors can indeed be chosen to be orthonormal as assumed. Using the equations 2.60 and 2.58 and making use of the orthonormality relation in equation 2.50, the value of the error criterion at the minimum is equal to

$$E_M = \frac{1}{N} \sum_{i=m+1}^{d} \lambda_i \qquad (2.61)$$

Thus, the minimum is obtained by choosing the $d-m$ smallest eigenvalues, and their corresponding eigenvectors, as the ones to discard.
The procedure described in this section is called *Karhunen-Loeve transformation* or *principal component analysis* (because the eigenvectors $u_i$ are called principal components).

### Intrinsic dimensionality




a) b)

Figure 2.19: Examples of two-dimensional data sets which can be well approximated by a single parameter $\eta$ ([Bishop, 1996, pp. 314–315])

Sometimes the dimensionality of a data set is lower than appears at the first glance. Two examples are shown in the figure 2.19. Consider the example a). The data points are represented by circles. Note that all the data points lie on a single circle. Although the data set is two-dimensional, it can be approximated

without loss of precision by a single parameter $\eta$, being in this case the angle. Example b) shows a noisy data set. Again, it is two-dimensional, but can be approximated by the parameter $\eta$, which describes the position of a data point on an imaginary curve.
The "real" dimensionality of such data sets is called *intrinsic dimensionality.* Linear dimensionality reduction techniques such as principal component analysis outlined previous section are incapable to discover this lower dimensionality.

**Neural networks for dimensionality reduction**

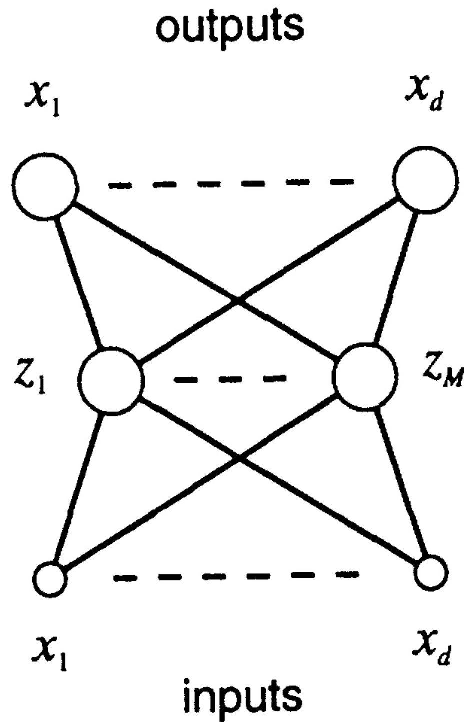



Figure 2.20: Auto-associative neural network for dimensionality reduction ([Bishop, 1996, p. 315]).

Auto-associative neural networks as shown in figure 2.20 can also be used for dimensionality reduction. In this case, the network is trained to map the $d$-dimensional input space onto itself "over" a $m$-dimensional ($m < d$) hidden layer. Thus, the input patterns fed to such a network are mapped onto them-

selves (input and target data are equal). If the hidden units have linear transfer functions, the error function has a unique global minimum and the principal components can be derived from the weight vector of the hidden layer (i.e. from the weights of connections from input to hidden layer). Such a network produces results comparable to the Karhunen-Loeve transformation.
According to Kramer [1991], the power of neural network as a principal component "analyst" can be extended by using 3 instead of 1 hidden layers. Such a network performs a non-linear principal component analysis and thus is capable of a better dimensionality reduction of the data.

### 2.2.6 Prior knowledge

In most cases, apart from the training and test data for neural networks, some general information about the data is available to the designers of a neural network. Such knowledge is called *prior knowledge*.
Consider, for example, a neural network, which should determine, whether a tissue is normal or a tumour (e.g. for the purpose of cancer screening). From medical statistics it may be known, that normal tissue is observed with a probability of 99 %, while tumours occur only in 1 % of observed data. But due to technical reasons (more efficient network training), the designers may choose to include equal amounts (50 % normal tissue, 50 % tumour) of normal and tumour examples in the training set. In order to be able to use such a (well-trained) network in a reality which significantly differs from the training data (difference between probabilities of observing normal and tumour tissue in training set and test set), additional measures may become necessary (see [Bishop, 1996, p. 319]).
The second problem that arises in the above example is the cost associated with a misclassification. Misclassifying a sick person as healthy has much severe consequences than misclassifying a healthy person as sick. This must also be taken into account in designing a neural network (see [Bishop, 1996, p. 319]).
Another type of prior knowledge are invariances. A classical example is a neural network used for shape recognition: it should be able to recognize a certain

shape irrespective of its rotation, location within the image or size (a rectangle remains a rectangle independent of whether it is rotated by 75 degrees or by 125).

There are three ways to incorporate invariances in a neural network:

1. Training the network by examples
2. Data preprocessing
3. "Hardwire" the invariance properties into the neural network

The first approach has no advantages but simplicity; such approach would require huge amounts of training data, which would be a problem in most cases. Continuing the shape recognition example, it would be necessary to train the network on a rectangle that is rotated by 0 degrees, $x$ degrees, $2x$ degrees etc (where $x$ is the step size and depends on precision).

A modification of this approach called *tangent prop* learns invariances from examples, but is trained in a different way and does *not* require increased amount of training data. All the approaches mentioned here are described in greater detail below.

### Tangent prop

Contrary[8] to the usual training data, where the networks were trained on input data and corresponding target data, in the case with tangent prop, not only the input and target values themselves are presented to the network, but it is also provided with information about how the variations of input data affect the target function. This information is expressed by means of partial derivations.

Consider a neural network that should learn the target function $f(x)$. It should take $x$ as input and produce $f(x)$ at the output. The normal approach is to use training data records of the structure $\langle \text{Input value}, \text{Target value} \rangle$, in this case $\langle x_i, f(x_i) \rangle$ ($i$ denotes the number of training sample). Tangent prop uses other training data, namely:

$$\langle \text{Input value,Target value,Change of target value with respect to change of input value} \rangle,$$

[8] This section is based upon [Mitchell, 1997, pp. 347–349]

in this case $\langle x_i, f(x_i), \frac{f(x_i)}{\partial x_i} \rangle$.

The advantage of providing the first derivative information along with the raw

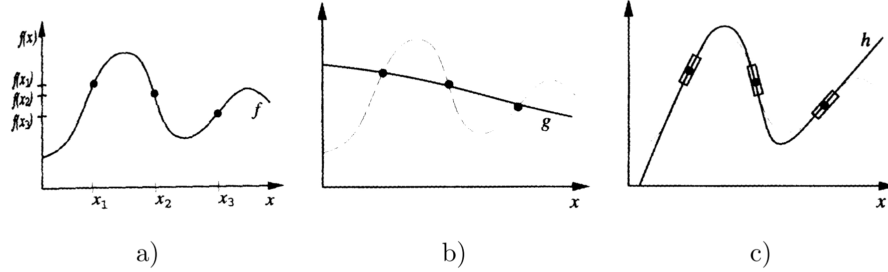



Figure 2.21: Fitting values and derivatives with tangent prop. Let $f$ be the target function for which three examples $x_1, x_2, x_3$ are known (a)). Based on these points the learner might generate the hypothesis $g$ (b)). If the derivatives are also known, the learner can state the more accurate hypothesis $h$ shown in figure c) ([Mitchell, 1997, p. 347]).

input data is illustrated in figure 2.21. The task of the learner is to learn the target function $f(x)$ using the training data that consists of the value $f(x)$ at three different points (a)). A "normal" neural network would learn the wrong hypothesis $g$, since this hypothesis is the one with minimal deviation from training data, the one with smallest residuals. But it is obviously wrong (b)). If, apart from the three values of $f(x)$, also the derivatives $\frac{f(x)}{\partial x}$ at these three points are provided, the situation changes crucially. Now the learner "knows" also the slopes (first derivative is the slope of a function at a certain point) of the function at these three points, shown in figure 2.21 (c) as rectangles. Having this information, a much more accurate approximation of $f(x)$, denoted by $h$ is possible.

Tangent prop is capable of learning any invariances, which can be expressed as differentiable functions. Apart from the data preprocessing, tangent prop has a slightly different transfer function. A detailed treatment of tangent prop can be found in Simard et al. [1992]

### Invariance through preprocessing

This method involves extraction of invariant features from the input data. In particular, the so-called *moments* and the already mentioned Fourier descriptors can be used for the particular task of shape recognition ([Bishop, 1996, pp. 322–324] and Seul et al. [2000]).

### Shared weights

This approach allows a neural network to recognize a shape correctly even if it is translated (i.e. its location within the image is changed.[9] Consider the network

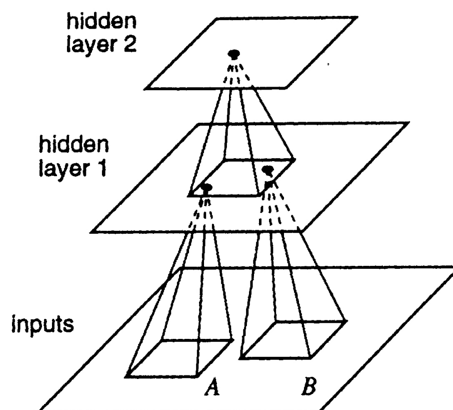



Figure 2.22: Schematic architecture of a network for transition-invariant object recognition in two-dimensional images. In a practical system there may be more than two layers between the input and the outputs ([Bishop, 1996, p. 325]).

architecture shown in figure 2.22. The inputs to the network are given by the intensities at each of the pixels in a two-dimensional array. Units in the first and second layers are similarly arranged in two-dimensional sheets to reflect the

[9]This is not the only approach to achieve translation invariance. Other (probably simpler) technique would firstly extract all the fragments of an image that may correspond to different shapes, and secondly, process each fragment individually by means of a neural network. In this case, the neural network doesn't need to be capable of translation invariant shape recognition. The process of extracting fragments from an image is called *segmentation* For a rigorous treatment of segmentation, see [Jain et al., 1995, pp. 73–87]

geometrical structure of the problem. Instead of having full interconnections between adjacent layers, each hidden unit receives inputs only from units in a small region in the previous layer, known as *receptive field*. This reflects the results of experiments in conventional image processing which have demonstrated the advantage of extracting local features from an image and then combining them together to form higher-order features. Note that it also imitates some aspects of the mammalian visual processing system. The network architecture is typically chosen so that there is some overlap between adjacent receptive fields. The technique of shared weights can then be used to build in some degree of translation invariance into the response of the network ([Bishop, 1996, p. 325]). In the simplest case this involves constraining weights from each receptive field to be equal to the corresponding weights from all of the receptive fields of the other units in the same layer. Consider an object which falls within receptive field shown at $A$ in figure 2.22, corresponding to a unit in hidden layer 1, and which produces some activation level in that unit. If the same object falls at the corresponding position in receptive field $B$, then, as a consequence of the shared weights, the corresponding unit in hidden layer 1 will have the same activation level. The units in the second layer have fixed weights chosen so that each unit computes a simple average of the activations of the units that fall within its receptive field. This allows units in the second layer to be relatively insensitive to moderate translations within the input image. However, it does preserve some positional information thereby allowing units in higher layers to detect more complex composite features. Typically, each successive layer has fewer units than previous layers, as information on the spatial location of objects is gradually eliminated. This corresponds to the use of a relatively high resolution to detect the presence of a feature in an earlier layer, while using a lower resolution to represent the location of that feature in a subsequent layer. The use of receptive fields can dramatically reduce the number of weights present in the network compared with a fully connected architecture. This makes it practical to treat pixel values in an image directly as inputs to a network. In addition, the use of shared weights means that the number of independent pa-

rameters in the network is much less than the number of weights, which allows much smaller data sets to be used than would otherwise be necessary. Shared weights networks have been used for the recognition of characters ([Bishop, 1996, p. 326]).

**Neural networks**

A special type of neural networks, so-called *higher-order networks* in combination with shared weights can be used for incorporating translation invariance into a neural network. Interested reader should refer to [Bishop, 1996, pp. 326–329].

## 2.3 Time series processing with neural networks

This section is based upon Dorffner [1996].

### 2.3.1 Overview

Neural networks, being developed primarily for the purpose of pattern recognition (classification), are not well suited for modelling time series because the original applications of neural networks were concerned with detection of patterns in arrays of measurements which do not change in time (Dorffner [1996]).The dynamic nature of spatio-temporal data (i.e. data that has a spatial and a temporal dimension) as time series are, requires introduction of additional mechanisms. In particular, a neural network used for time series processing must possess memory in one way or the other.
One way to supply the neural network with memory is to use a *time window*. In this case, a certain number of past time series elements is provided to the network as inputs and the network produces a prediction of the next element. This approach is discussed in section 2.3.2.
Another way to provide memory to the neural network is to store past values of output (context layer) or hidden (state layer) nodes in additional layers. These additional layers are connected to the hidden layer in a similar way as the input

layer. Networks implementing these approaches are treated in sections 2.3.4 and 2.3.5.

### 2.3.2 Multi-layer perceptron

Multi layer perceptron neural networks can be used for time series processing and are similar to the autoregressive (AR) process described in section 1.4.1 (Dorffner [1996]). Suppose a certain function $f(\vec{y}) : \Re^n \rightarrow \Re^m$ has to be approximated by a MLP neural network. Formally, the approximation given by the neural network is equal to (Dorffner [1996])

$$\hat{f}_{MLP}(\vec{y}) = \left( \sum_{j=1}^{k} v_{jl} \,\text{transfer} \left( \sum_{i=1}^{n} w_{ij} p_i - \theta_j \right) - \theta_l \right), l = 1, 2, \ldots, m \quad (2.62)$$

where transfer is the transfer function, $k$ is the number of hidden units, $v_{jl}$ and $w_{ij}$ are weights and $\theta_i$ are threshold values.
If such a network has the architecture shown in figure 2.23 where the individual input nodes are "connected" to the elements of the time series $1, 2, \ldots, p$ steps in the past, it can be viewed as a special case of the AR model. This statement will be explained in more detail below.
Every autoregressive model has the property that it approximates the time series by applying a certain function to several elements of the time series in the past:

$$\text{AR}(p) : \hat{y}_t = f(y_{t-1}, y_{t-2}, \ldots, y_{t-p}) + \varepsilon_t \quad (2.63)$$

In the "classical" AR model discussed on p. 25, the function $f(y_{t-1}, y_{t-2}, \ldots, y_{t-p})$ was equal to

$$f(y_{t-1}, y_{t-2}, \ldots, y_{t-p}) = \mu + \left( \sum_{i=1}^{p} \gamma_i y_{t-1} \right) \quad (2.64)$$

However, it is possible to take another function instead of this linear function used in the "classical" AR model, for example, the MLP approximation function given in equation 2.62. Provided that the architecture is the one shown in figure 2.23, the MLP neural network works very similar to the classical AR model with the only difference that it is more powerful due to the nonlinear function

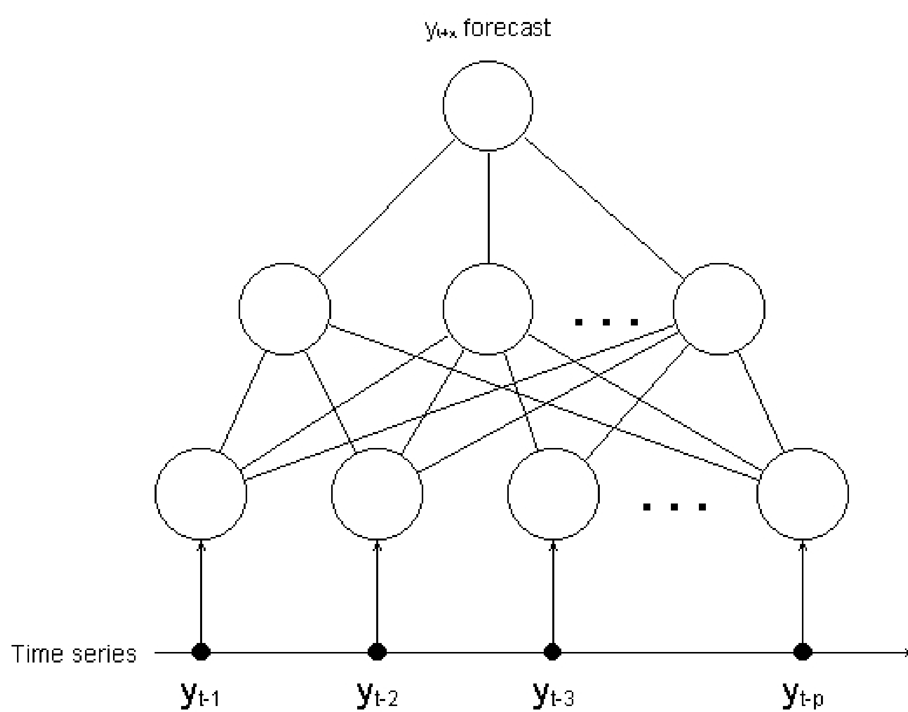



Figure 2.23: A feedforward neural network with time window as a non-linear AR model (Dorffner [1996], [Bishop, 1996, p. 303])

$f(y_{t-1}, y_{t-2}, \ldots, y_{t-p})$. Seen from this point of view, a neural network is a non-linear AR model. Compared to classical AR models, such a neural network has several advantages (Dorffner [1996]:

- A neural network can model much more complex underlying characteristics of the series
- A neural network theoretically do not have to assume stationarity.

However, the neural network approach also has some disadvantages when compared to classical AR model (Dorffner [1996]):

- Neural networks require large numbers of sample data, due to their large number of parameters (weights)
- Neural networks can run into a variety of problems such as overfitting, capture in local minima etc.
- do not necessarily include the linear case in a trivial way.

Due to the first problem, neural network approaches in real-world applications are often ruled out in spite of a non-linear data, since training data is not available to a sufficient degree. The second point addresses the choice of the learning algorithm. Backpropagation is often not the most appropriate choice to obtain optimal results. Examples of neural networks of this type used for time series prediction purposes can be found in Chakraborty et al. [1992], Refenes et al. [1991], Weigend et al. [1990] and White [1993] (Dorffner [1996]).

### 2.3.3 Time-delay neural network

Another mechanism to supply neural networks with "memory" to deal with the temporal dimension is the introduction of time delays on connections. In other words, through delays, inputs arrive at hidden units at different points in time, thus being "stored" long enough to influence subsequent inputs. This approach, called a *time-delay neural network* (TDNN) has been extensively employed in speech recognition (e.g. Waibel [1989]). Formally, time delays are identical to

time windows and can thus be viewed as autoregressive models as well. An interesting extension is the introduction of time delays also on connections between hidden and output units, providing additional, more "abstract" memory to the network (Dorffner [1996]).

### 2.3.4 Jordan networks

Figure 2.24 shows another approach for processing time series using neural networks, the so-called *Jordan* network. It is a multi-layer perceptron with one hidden layer and a feedback loop from the output layer to an additional input layer called *context layer*. Each node in the context layer is connected to itself via self-recurrent loops with a weight smaller than 1 (Dorffner [1996]).
Hence, a Jordan network takes into account not only past time series elements, but also its own forecasts. This property has often given rise to the argument that recurrent networks can exploit information beyond a limited time window. However, in practice this cannot really be exploited. If the weight of a connection to a context node is close to 1, the node (if it uses a sigmoid transfer function) quickly saturates to maximum activation, where additional inputs have little effect. If the weight is very small in comparison to 1, the influence of past estimates quickly goes to 0 (Dorffner [1996]).
The Jordan networks can be seen as an extension of the ARMA models discussed in section 1.4.1 (Dorffner [1996]). In spite of the advantages of this approach (potentially higher power), there also exist limitations for applicability of Jordan networks to real-world time series already mentioned above (Dorffner [1996]).

### 2.3.5 Elman networks

An Elman network (figure 2.25) possesses an additional layer called *state layer*, by means of which the outputs of the hidden nodes are fed back to the network. Although the Elman network in the original form has only limited modelling capabilities, it can be extended to perform as a universal state-space model[10].

[10] A state space model describes (and forecasts) time series under the assumption that the next element of the time series can be predicted by the state the system currently is in, no

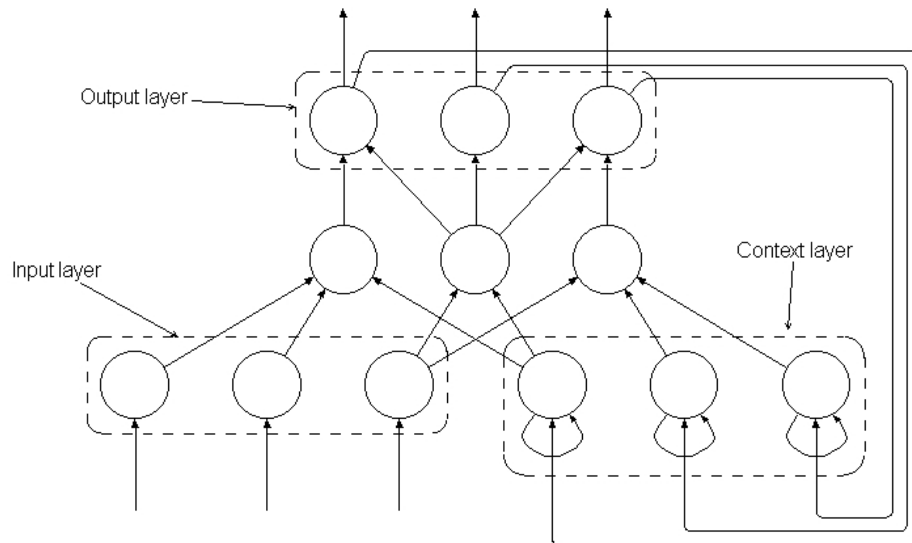



Figure 2.24: Jordan network (Dorffner [1996])

Real-world applications of Elman networks are described in Gordon et al. [1991], Dorffner et al. [1994] and Debar and Dorizzi [1992].

### 2.3.6 Multi-recurrent networks

Ulbricht [1994] and Ulbricht [1995] has given an extensive overview of additional types of recurrencies, time-windows and time delays in neural networks. By combining several types of feedback and delay one obtains the general *multirecurrent network* (MRN), depicted in figure 2.26 (Dorffner [1996]).

This type of neural network has several remarkable properties. Firstly, feedback from hidden *and* output layers are permitted. Secondly, all input layers (the actual input, the state and the context layer) are permitted to be extended by time-delays, such as to introduce time windows over past instances of the corresponding vectors. Thirdly, like in the Jordan network, self-recurrent loops in the state layer can be introduced. The weights of these loops, and the weights of the feedback copies resulting from the recurrent one-to-one connections, are

matter how the state was reached. All the history of the series necessary for producing the next element can be expressed by one state vector (Dorffner [1996]).

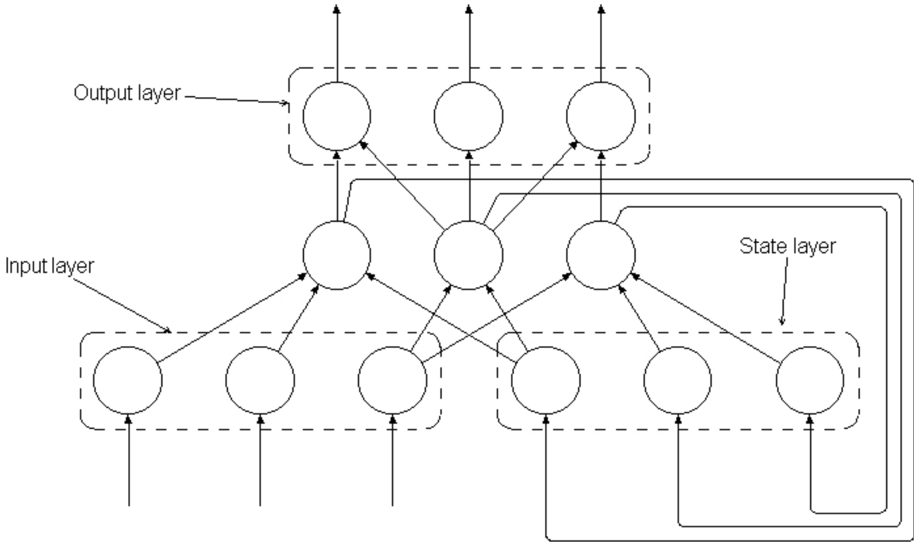



Figure 2.25: An example of an Elman network (Dorffner [1996])

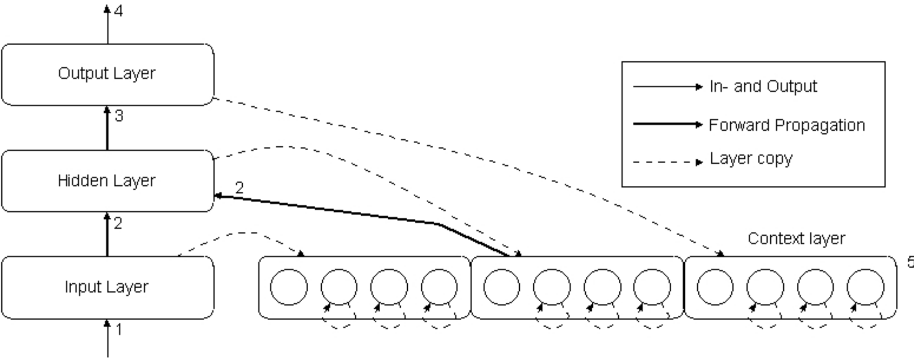



Figure 2.26: A multi-recurrent network (Ulbricht [1995], Dorffner [1996])

chosen such as to scale the theoretically maximum input to each unit in the state layer 1, and to give more or less weight to the feedback connections of the self-recurrent loops, respectively (Dorffner [1996]).

If, for example, 75 % of the total activation of a unit in the state layer comes from the hidden layer feedback, and 25 % comes from self-recurrency, the state layer will tend to change considerably at each time step. If, on the other hand, only 25 % come from the hidden layer feedback, and 75 % from the self-recurrent loops the state vector will tend to remain similar to the one at the previous time step (Dorffner [1996]).

Ulbricht [1995] speaks of *flexible* and *sluggish* state spaces, respectively. By introducing several state layers with different such weighting schemes, the network can exploit both the information of rather recent time steps and a kind of average of several past time steps, i.e. a longer, averaged history (Dorffner [1996]).

It is clear that a full-fletched version of the MRN contains a very large number of parameters (weights) and requires even more care than the other models discussed above. Several empirical studies (Ulbricht [1994], Ulbricht et al. [1996]) have shown, however, that for real-world applications, some versions of the MRN can significantly outperform most other, more simple, forecasting methods. The actual choice of feedback, delays and weightings still depends largely on empirical evaluations, but there are iterative algorithms similar to one used for determination of the parameters of the ARMA model (figure 1.7, p. 28) that appear applicable to this type of network (Dorffner [1996]).

Another advantage of self-recurrent loops becomes evident in applications where patterns in the time series can vary in time scale. This phenomenon is called *time warping*, and is especially known in speech recognition, where different speech patterns can vary in length and relationships between segments dependent on speaking speed and intonation (Morgan and Scotfield [1991]). In autoregressive models with fixed time windows, such distorted patterns lead to vectors that do not share sufficient similarities to be classified correctly. This is sometimes called the *temporal invariance problem* - the problem of recognizing temporal

patterns independent of their duration and temporal distortion. In a recurrent neural network with self-recurrent loops such invariances can be dealt with, especially when sluggish state spaces are employed. If state vector is forced to be similar at subsequent time steps, events can be treated equally (or similarly) even when they are shifted along temporal dimension (Dorffner [1996]).

### 2.3.7 Other network architectures

While the most important neural network approaches to time series processing have been described in previous sections, there exists a variety of many other design approaches, several of which are listed below (Dorffner [1996]):

- Many time series applications are tackled with *fully recurrent networks*, or networks with recurrent architectures different from the ones already discussed (e.g. Rementeria et al. [1994]). Special learning algorithms for arbitrary recurrent networks have been devised, such as *backpropagation in time* (Rumelhart et al. [1986]) and real-time recurrent learning (Williams and Zipser [1989], Dorffner [1996]).

- Many authors use a combination of neural networks with so-called *hidden Markov models* (HMM) for time series and signal processing. HMMs are related to finite state automata and describe probabilities for changing from one state to the other. Detailed treatment of this topic can be found in Bourlard and Morgan [1989] and Bengio [1995](Dorffner [1996]).

- Unsupervised neural network learning algorithms, such as the self-organizing map, can also be applied in time series processing, both in forecasting (Baumann and Germond [1993]) and classification (Roberts and Tarassenko [1992]). The latter application constitutes an instance of so-called *spatio-temporal clustering*, i.e. the unsupervised classification of time series into clusters - in this case the clustering of sleep-EEG into sleep stages (Dorffner [1996]).

- A number of authors have investigated the properties of neural networks viewed as dynamical systems including chaotic attractor dynamics (Kolen

[1994], Port et al. [1995]).

# Chapter 3

# Financial time series prediction and ANNs

## 3.1 Historical Background

The history[1] of neural network development can be divided into five main stages, spanning over 150 years. These stages are shown in figure 3.1, where key research developments in computing and neural networks are listed along with evidence of the impact these developments had on the business community. Much of the preliminary research and development was achieved during stage 1 which here is considered to be pre-World War II (i.e. prior to 1945). During this time most of the foundations for future neural network research had been formed. The basic design principles of analytic engines had been invented by Charles Babbage in 1834, which became the forerunner to the modern electronic computer. The ability of these analytic engines and adding machines to automate tedious calculations led to their widespread use by 1900 (the US government used such machines for the 1890 national census), and International Business Machines (IBM) was founded in 1914 to capture this market. Meanwhile researchers in psychology had been exploring the human brain and learning. William James'

[1] This section is based upon explanations in Smith and Gupta [2000].

1890 book *Psychology* discussed some of the early insights researchers had into the nature of brain activity.

Between the two World Wars, Alan Turing investigated computing devices which used the human brain as a paradigm, and the field of artificial intelligence was born. This first stage of preliminary research concludes with the first basic attempts to mathematically describe the workings of the human brain. McCulloch and Pitts' (1943) paper entitled *A logical calculus of the ideas immanent in nervous activity* proposed a simple neuron structure with weighted inputs and neurons which are either "on" or "off". At this stage, however, these neural networks could not learn, and the lack of suitable computing resources stifled experimentation.

Stage 2 is characterised by the age of computer simulation. In 1946, Wilkes designed the first operational stored-program computer. Over the ensuing years, the development of electronic computers progressed rapidly, and in 1954 *General Electric Company* became the first corporation to use a computer when they installed a UNIVAC I to automate the payroll system. The advances in computing enabled neural network researchers to experiment with their ideas, and in 1949 Donald Hebb wrote *The Organization of Behaviour*, where he proposed a rule to allow neural network weights to be adapted to reflect the learning process explored by Pavlov. In 1954, Marvin Minsky built the first neurocomputer based on these principles. In the summer of 1956, the Dartmouth Summer Research Project was held and attracted the leading researchers at the time. The field of neural networks was officially launched at this meeting. Rosenblatt's Perceptron model soon followed in 1957, and many simple examples were used to show the learning ability of neural networks. By this stage the fields of artificial intelligence and neural networks were causing much excitement amongst researchers, and the general public was soon to become captivated by the idea of "thinking machines". In 1962, Bernard Widrow appeared on the US documentary program *Science in Action* and showed how his neural network could learn to predict the weather, blackjack, and the stock market. For the remainder of the 1960s this excitement continued to grow.

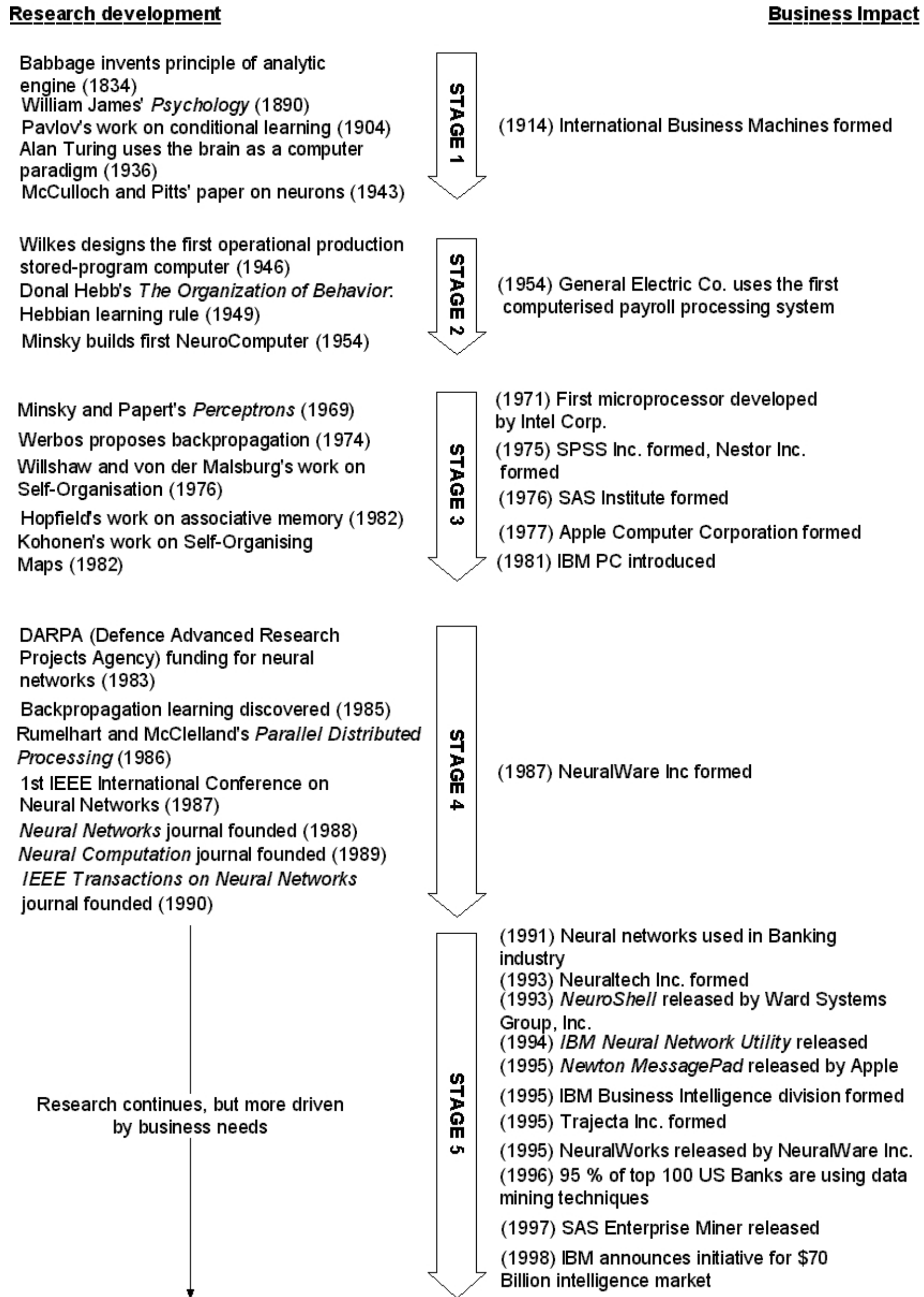



Figure 3.1: The five stages of neural network research development and its business impact (Smith and Gupta [2000])

Then in 1969 a book was published which severely dampened this enthusiasm. The book was Minsky and Papert's *Perceptrons* (Minsky and Papert [1969]), that proved mathematically that perceptrons are incapable of learning any problem containing data that are linearly inseparable. The consequence of their book was that much neural network research ceased. This is the third stage, commonly called the "quiet years" a from 1969 until 1982. During this time, however, there were significant developments in the computer industry. In 1971 the first microprocessor was developed by the Intel Corporation. Computers were starting to become more common in businesses worldwide, and several computer and software companies were formed during the mid-seventies. SPSS Inc. and Nestor Inc. in 1975 and Apple Computer Corporation in 1977 are a few examples of companies which formed then, and later became heavily involved in neural networks. In 1981, IBM introduced the IBM PC which brought computing power to businesses and households across the world. While these rapid developments in the computing industry were occurring, some researchers started looking at alternative neural network models which might overcome the limitations observed by Minsky and Papert. The concept of self-organisation in the human brain and neural network models was explored by Willshaw and von der Malsburg, and consolidated by Kohonen in 1982. This work helped to revive interest in neural networks, as did the efforts of Hopfield who was looking at the concepts of storing and retrieving memories. Thus, by the end of this third stage, research into neural networks had diversified, and was starting to look promising again.

From 1983 until 1990 marks the 4th stage where neural network research blossomed. In 1983 the US government funded neural network research for the first time through the Defence Advanced Research Projects Agency (DARPA), providing testament to the growing feeling of optimism surrounding the field. An important breakthrough was then made in 1985 which impacted on the future of neural networks considerably. Backpropagation was discovered independently by two researchers which provided a learning rule for neural networks which overcame the limitations described by Minsky and Papert. In actual fact, back-

propagation had been proposed by Werbos while he was a graduate student some 10 years earlier, but remained undiscovered until after LeCun and Parker had published their work. The backpropagation algorithm enabled any complex problem to be learnt without the limitations of Perceptrons. Within years of its discovery the neural network field grew dramatically in size and momentum. Rumelhart and McClelland's (1986) book, "Parallel Distributed Processing", became the neural network "bible". In 1987, the Institute of Electrical and Electronic Engineers (IEEE) held the 1st International Conference on Neural Networks, and these conferences have been held annually ever since. Many neural network journals emerged over the next few years, with notable ones being *Neural Networks* in 1988, *Neural Computation* in 1989, and *IEEE Transactions on Neural Networks* in 1990. During this stage of rapid growth, the business world remained fairly untouched by neural networks. A few companies specialising in neural networks formed such as NeuralWare Inc. in 1987, and the reputation of neural networks in the business community was beginning to grow, but it was not until the next stage that neural network made their real and lasting impact in business. In 1991, the banks started to use neural networks to make decisions about loan applicants and speculate about financial prediction. This marks the start of the 5th stage. Within a couple of years many neural network companies had been formed including Neuraltech Inc. in 1993 and Trajecta Inc. in 1995. Many of these companies produced easy-to-use neural network software containing a variety of architectures and learning rules. A survey of neural network software products available in 1993 listed over 50 products, the majority of which were designed to be run on a PC under Microsoft Windows. The impact on business was almost instantaneous. By 1996, 95% of the top 100 banks in the US were utilising intelligent techniques including neural networks. Within competitive industries like banking, finance, retail, and marketing, companies realised that they could use these techniques to help give them a "competitive edge". In 1998, IBM announced a company-wide initiative for the estimated $70 billion business intelligence market. Research during this 5th stage still continues, but it is now more industry driven. Now that the busi-

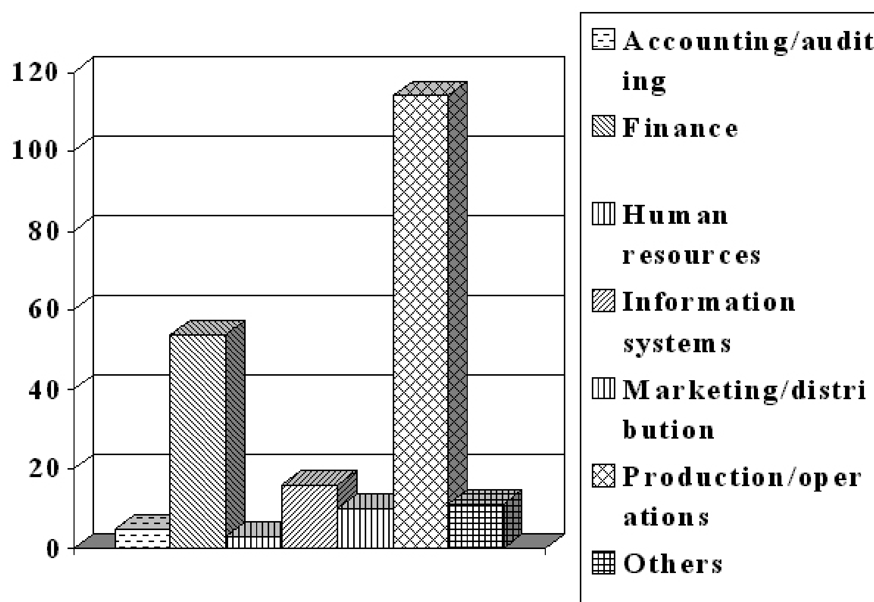



Figure 3.2: Application domains of neural networks (Wong et al. [1997a])

ness world is becoming increasingly dependent upon intelligent techniques like neural networks to solve a variety of problems, new research problems are emerging. Researchers are now devising techniques for extracting rules from neural networks, and combining neural networks with other intelligent techniques like genetic algorithms, fuzzy logic and expert systems. As more complex business problems are tackled, more research challenges are created.

## 3.2 Application of neural networks in today's business

While the previous section outlined the past of neural networks in business, this section describes the present day of ANNs. According to Wong et al. [1997a], the most frequent application domains of ANN are productions/operations (53.5 %) and finance (25.4 %; Wong et al. [1997a], Zekic [1998], figure 3.2) According to Wong et al. [1997b], approximately 95 % of reported neural network business application studies utilise multilayered feedforward neural networks with the

backpropagation learning rule (MLP/EBP[2]).

Apart from MLP/EBP, Hopfield networks and self-organizing maps are of practical importance in the financial domain (Smith and Gupta [2000]).

According to Zekic (Zekic [1998]), who analysed various ANNs in the business domain, ANNs used for

- predicting stock performance
- recommendation for trading
- classification of stocks
- predicting price changes of stock indexes
- stock price prediction
- modeling the stock performance
- forecasting the performance of stock prices

dominate the research. In most analyzed applications, the NN results outperform statistical methods, such as multiple linear regression analysis, discriminative analysis and others (Zekic [1998]).

## 3.3 Examples of ANNs in finance

### 3.3.1 Example 1

The first example of the application of neural networks to financial time series prediction is the system developed by Tino et al. [2000].

The system described is used for simulation of option trading with FTSE and DAX options. The options are traded using a delta-neutral trading strategy, i.e. the profit of a trading action (buying or selling options) depends on correct prediction of the volatility of the FTSE and DAX index in future. Thus, the system predicts the volatility.

[2]MLP/EBP refers to a multi-layer perceptron trained with stochastic gradient descent backpropagation learning algorithm presented in section 2.1.6.

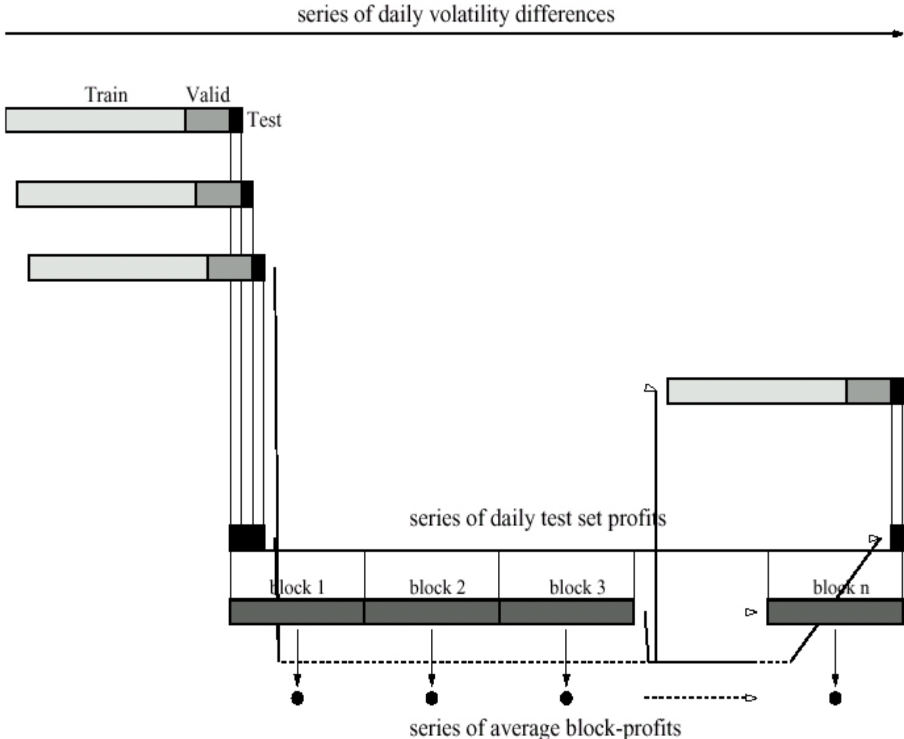



Figure 3.3: Sliding window technique used in the experiment of Tino et al. [2000]

In order to accommodate for changing properties of the time series, a sliding window technique is applied (see figure 3.3). The neural networks and some other algorithms are trained on the training set of 500 daily volatility values. Then, every algorithm is applied on a validation set of 125 days, i.e. the algorithms are used to "trade" with options in these 125 days. Finally, the best performing algorithm is chosen and applied to a test set of 5 days. Then, the sliding window is shifted by 5 days and the process begins again (only the profits on the test sets are counted towards overall profit of the prediction system).
There are several prediction techniques incorporated in the system, among them two neural networks. Both are trained on 1, 3, 6 or 10 past volatility differences. One is used to predict the volatility difference for the next day and the other to predict the direction of the volatility difference (up or down).

The neural networks outperform the GARCH model in most cases. However, the most interesting result of the study of Tino et al. [2000] is the performance of their so-called *Simple class*:

> Our idea is to use yet another model class, Simple, that is a small collection of simple, fixed predictors requiring no training on the training set. A suitable candidate model to be applied on the test set is selected on the validation set. The class Simple avoids using the old data in the sliding window. Of course, the price to pay is the fixed nature of the models in Simple. However, in financial prediction tasks, simple, short memory models often outperform more sophisticated predictors. Our choice of models for the class Simple is a collection of four simple-minded predictors operating on the series of volatility differences quantized using the two-symbol scheme: always predict 1 (decrease), always predict 2 (increase), copy the last symbol and revert the last symbol (i.e. predict the other symbol).

This simple-minded model class outperforms all other prediction techniques applied in this experiment (Markov models, variable length Markov models, fractal classification machines, neural networks, GARCH models):

> As typical in financial prediction tasks, simple-minded predictors in the class Simple are difficult to beat by more complicated models and should definitely be considered in studies like this one.

It is noticeable that in Tino et al. [2000] the profit after subtraction of transaction costs is taken into account. The conclusion of such results is that prediction accuracy (which is certainly better at neural networks and GARCH models than in the Simple class) does not always guarantee a profitable trading. It appears that prediction accuracy and profitability of the trading strategy predictions are two different tasks.

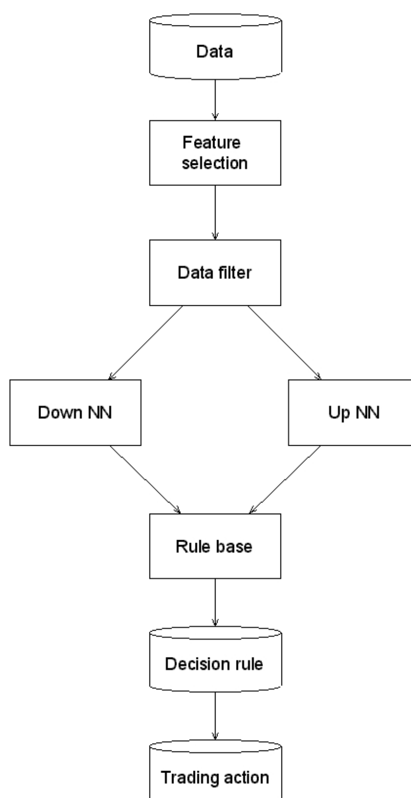



Figure 3.4: Structure of the system described in Chenoweth and Obradovic [1996]

### 3.3.2 Example 2

The second example of ANNs's application in finance is the experiment described in Chenoweth and Obradovic [1996].

This paper investigates the behaviour of a system for prediction of the S&P 500 index. The system structure is given in figure 3.4. The feature selection happens by means of sequential search techniques (see p. 68) and a ranking method which determines the best feature subset. The "quality" of feature subset is expressed in terms of class separation. Chenoweth and Obradovic [1996] use Euclidean, Patrick-Fisher, Mahalanobis and Bhattacharyya distances for measuring class separation. All of these measures are - with respect to the purpose they serve - equivalent to the Fisher's criterion presented in appendix I

and defined in equation I.36 on page 187.

The preprocessed data are passed to the data filter. It reduces noise and splits the data to the two neural networks. The noise reduction is done by discarding the data record (pattern), if the absolute return (returns are part of each pattern, see p. 21) is smaller than a certain threshold value. Thus, small changes are treated as noise and discarded.

Patterns which are not discarded are used to train the neural networks. If the corresponding return ("raw", not absolute return) is greater than the threshold value, it is fed to the "up" ANN, if it is lower than ($(-1) \cdot$ threshold value), then it is fed to the "down" ANN.

In order to accommodate for non-stationarity, the sliding window technique is used in this study, too. Unlike the one in Tino et al. [2000] who use a step size of 5, the step size used in the study of Chenoweth and Obradovic [1996] is 1.

When trained, the neural networks are used to form a return prediction for the next day. These two predictions are used as input for the rule base, which then generates a buy or sell recommendation.

The important outcome of this study is the fact that dual-ANN approach outperforms a technique, where only one ANN is used. Furthermore, the efficient preprocessing seems to play an essential role in efficient prediction formation. Using their ranking feature selection algorithm, they extract few (6 to 10) features out of a pool of up to 30 candidate features.

### 3.3.3 Example 3

The last example will be a neural network presented in Terna [2000].

In this experiment, several neural networks are used to simulate the behaviour of a small economic system, consisting of 20 participants (10 buyers and 10 sellers). Equipped with a minimal set of rules and a neural network for learning to adapt to the environment, these agents behave in the following way [Terna, 2000, p. 5]:

> "At every simulation step (i.e., a tick of the simulation clock), artificial consumers look for a vendor; all the consumers and vendors

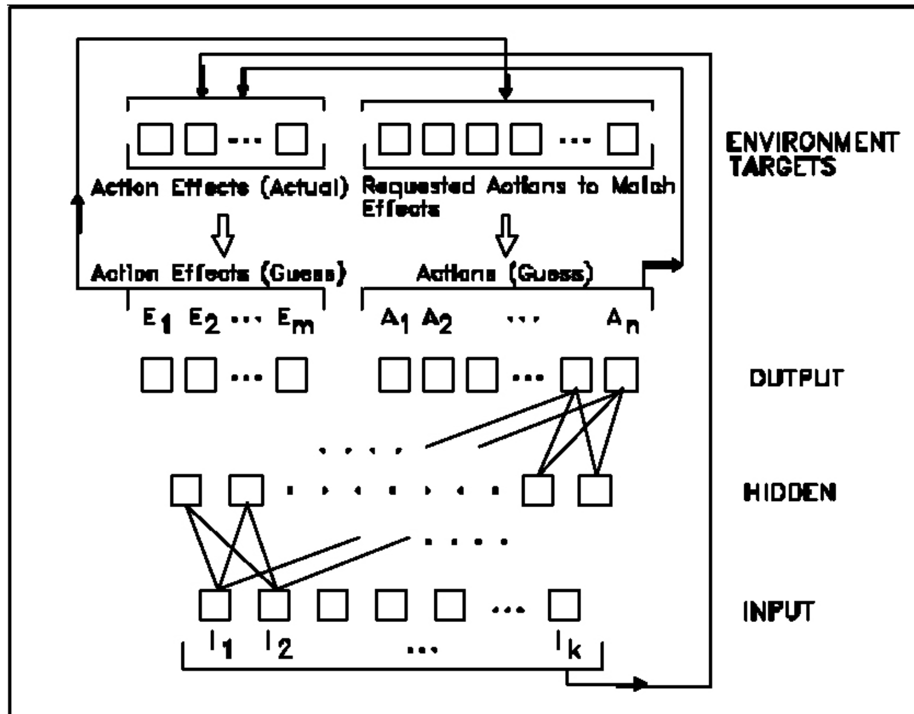



Figure 3.5: Design of the neural network in the agents presented in the study of Terna [2000]

are randomly matched at each step. An exchange occurs if the price asked by the vendor is lower than the level fixed by the consumer. If a consumer has not been buying for one or more than one step, it raises its price level by a fixed amount according to the counter rule and the sensitivity parameter introduced below. It acts in the opposite way if it has been buying and its inventory is greater than one unit. A simulated vendor behaves in a symmetric way (but without a sensitivity parameter): it chooses the offer price randomly within a fixed range. If the number of steps for which it has not been selling is greater than one, it decreases the minimum and maximum boundaries of this range, and vice versa if it has been selling."

The neural networks used in the agents (sellers and buyers) have two output types

1. actions to be performed and

2. guesses about the effects of those actions.

Both the targets necessary to train the network from the point of view of the actions and those connected with the effects are built in a crossed way, originating the name *Cross Targets* (CT). The former are built in a consistent way with the outputs of the network concerning the guesses on the effects, in order to develop the capability to decide actions close to the expected results. The latter are similarly built with the outputs of the network concerning the guesses of the actions, in order to improve the agent's capability of estimating the effects emerging from the actions that the agent herself is deciding. The method of CTs, introduced to develop economic subjects' autonomous behaviour, can also be interpreted as a general algorithm useful for building behavioral models without using constrained or unconstrained optimization techniques. The kernel of the method, conveniently based upon artificial neural networks, is learning by guessing and doing: control capabilities of the subject can be developed without defining either goals or maximizing objectives. Figure 3.5 describes a CT agent learning and behaving in a CT scheme. The agent has to produce guesses about its own actions and related effects, on the basis of an information set (the input elements are $I_1, \ldots, I_k$). Remembering the requirement of internal consistency, targets in learning process are:

1. on one side, the actual effects - measured through accounting rules - of the actions made by the simulated subject;

2. on the other side, the actions needed to match guessed effects. In the last case we have to use inverse rules, even though some problems arise when the inverse function is undetermined.

The field of *agent based computational finance* is a highly interesting and actively researched area. Of all ANN application in finance, it is perhaps the one which

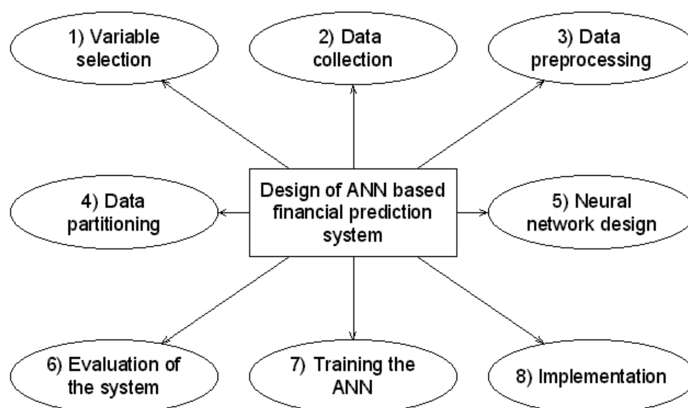



Figure 3.6: Design process of a neural network based financial time series prediction system (Kaastra and Boyd [1996]).

can provide the deepest insights into the fundamental nature of the economy. For a rigorous treatment of application of neural networks in this context, refer to Beltratti et al. [1996].

## 3.4 Design of ANNs in finance

The methodology described in scope of this section is based upon Kaastra and Boyd [1996]. The design of a neural network successfully predicting a financial time series is a complex task. The individual steps of this process are shown in figure 3.4 and listed below:

1. Variable selection
2. Data collection
3. Data preprocessing
4. Data partitioning
5. Neural network design
6. Evaluation of the system
7. Training the ANN

8. Implementation

Detailed description of individual steps is given below.

### 3.4.1 Step 1: Variable selection

Success in designing a neural network depends on a clear understanding of the problem (Nelson and Illingworth [1991]). Knowing which input variables are important in the market being forecasted is critical. This is easier said than done because the very reason for relying on a neural network is for its powerful ability to detect complex nonlinear relationships among a number of different variables. However, economic theory can help in choosing variables which are likely important predictors. At this point in the design process, the concern is about the raw data from which a variety of indicators will be developed. These indicators will form the actual inputs to the neural network (Kaastra and Boyd [1996]).

The financial researcher interested in forecasting market prices must decide whether to use both technical and fundamental economic inputs from one or more markets. Technical inputs are defined as lagged[3] values of the dependent variable[4] or indicators calculated from the lagged values. Fundamental inputs are economic variables which are believed to influence the dependent variable. The simplest neural network model uses lagged values of the dependent variable(s) or its first difference as inputs. Such models have outperformed traditional ARIMA-based models in price forecasting, although not in all studies (Sharda and Patil [1994], Tang et al. [1990]). A more popular approach is to calculate various technical indicators which are based only on past prices of the market being forecasted (Deboeck and Cader [1994]). As an additional improvement, intermarket data can be used since the close link between all kinds of markets, both domestically and internationally, suggests that using technical inputs from a number of interrelated markets should improve forecasting

[3] "Lagged" means an element of the time series in the past. For example, at time $t$, the values $y_{t-1}$,$y_{t-2}$,$y_{t-p}$ are said to be lagged values of the time series $y$.

[4] Dependent variable is the variable whose behavior should be modelled or predicted ([Dougherty, 1992, p. 54]).

performance. For example, intermarket data such as the Deutsche Mark/Yen and Pound cross rates and interest rate differentials could be used as neural network inputs when forecasting the D-Mark. Fundamental information such as the current account balance, money supply or wholesale price index may also be helpful (Kaastra and Boyd [1996]).

The frequency of the data depends on the objectives of the researcher. A typical off-floor trader in the stock or commodity futures markets would likely use daily data if designing a neural network as a component of an overall trading system. An investor with a longer term horizon may use weekly or monthly data as inputs to the neural network to formulate the best asset mix rather than using a passive buy and hold strategy. An economist forecasting the gross domestic product (GDP), unemployment or other broad economic indicators would likely use monthly or quarterly data (Kaastra and Boyd [1996]).

### 3.4.2 Step 2: Data collection

The researcher must consider cost and availability when collecting data for the variables chosen in the previous step. Technical data is readily available from many vendors at a reasonable cost whereas fundamental information is more difficult to obtain. Time spent collecting data cannot be used for preprocessing, training and evaluating network performance. The vendor should have a reputation of providing high quality data; however, all data should still be checked for errors by examining day to day changes, ranges, logical consistency (e.g. high greater than or equal to close, open greater or equal to low) and missing observations (Kaastra and Boyd [1996]).

Missing observations which often exist, can be handled in a number of ways. All missing observations can be dropped or a second option is to assume that the missing observations remain the same by interpolating or averaging from nearby values. Dedicating an input neuron to the missing observations by coding it as a one if missing and zero otherwise is also often done (Kaastra and Boyd [1996]).

When using fundamental data as an input in a neural network four issues must be kept in mind. First, the method of calculating the fundamental indicator

should be consistent over the time series. Second, the data should not have been retroactively revised after its initial publication as is commonly done in databases since the revised numbers are not available in actual forecasting. Third, the data must be appropriately lagged as an input in the neural network since fundamental information is not available as quickly as market quotations. Fourth, the researcher should be confident, that the source will continue to publish the particular fundamental information or other identical sources are available (Kaastra and Boyd [1996]).

### 3.4.3 Step 3: Data preprocessing

As in most other neural network applications, data preprocessing is crucial for achieving a good prediction performance when applying neural networks for financial time series prediction. The input and output variables for which the data was collected are rarely fed into the network in raw form. At the very least, the raw data must be scaled between the upper and lower bounds of the transfer functions (usually between zero and one or minus one and one; Kaastra and Boyd [1996]).

Two of the most common data transformations in both traditional and neural network forecasting are first differencing and taking log[5] of a variable. First differencing, or using changes in a variable, can be used to remove a linear trend from the data. Logarithmic transformation is useful for data which can take on both small and large values. Logarithmic transformations also convert multiplicative or ratio relationships to additive which is believed to simplify and improve the network training (Masters [1993]; Kaastra and Boyd [1996]).

Another popular data transformation is to use ratios of input variables. Ratios highlight important relationships (e.g. hog/corn, financial statement ratios) while at the same time conserving degrees of freedom because fewer input neurons are required to code the independent variables (Kaastra and Boyd [1996]).

Besides first differences, logs and ratios, technical analysis can provide a neural network with a wealth of indicators including a variety of moving averages, os-

[5]"log" means "logarithm".

cillators, directional movement and volatility filters. It is a good idea to use mix of different indicators to reduce variable redundancy and provide network with the ability to adapt to changing market conditions through periodic retraining (Kaastra and Boyd [1996]).

Smoothing both input and output data by using either simple or exponential moving averages is often employed. Empirical work on testing the efficient market hypothesis has found that prices exhibit time dependency or positive autocorrelation while price changes around a trend are somewhat random (Tomek and Querin [1984]). Therefore, attempting to predict price changes around the trend by using either unfiltered prices or price changes as inputs may prove to be difficult. Using moving averages to smooth the independent variables and forecasting trends may be a more promising approach (Kaastra and Boyd [1996]).

Sampling or filtering of data refers to removing observations from the training and testing sets to create a more uniform distribution. The type of filtering employed should be consistent with the objectives of the researcher. For example, a histogram of price changes for a commodity would reveal many small changes from which an off-floor speculator cannot profit after deducting realistic execution costs. However, this dense region of the distribution will greatly impact the training of the neural network since small price changes account for the majority of the training facts. The network minimizes the sum of squared errors (or other error function) over all the training facts. By removing these small price changes, overall trading performance can be improved since the network specializes on the larger, potentially profitable price changes. It is possible for trading systems to be unprofitable even if the neural network predicted 85 % of the turning points, as the turning points may be only small unimportant price changes (Deboeck [1994]). On the other hand, a floor trader holding positions overnight is likely interested in these small price changes. The researcher must be clear on what exactly the neural network is supposed to learn. Another advantage of filtering is a decrease in the number of training facts which allows testing of more input variables, random starting weights, or hidden neurons rather than training large data sets (Kaastra and Boyd [1996]).

In practice, data preprocessing involves much trial and error. One method to select appropriate input variables is to test various combinations. For example, a "top 20" list of variables consisting of a variety of technical indicators could be pretested ten at a time with each combination differing by two or three variables. Although computationally intensive, this procedure recognizes the likelihood that some variables may be excellent predictors only when in combination with other variables. Chaos theory and statistical tests cannot make such a determination. Also, the top 20 list can be modified over time as the researcher gains experience on the type of preprocessing that works for his/her application. This approach is especially useful if the training set is small relative to the number of parameters (weights) which is likely the case if all 20 input variables are presented to the neural network at once (Kaastra and Boyd [1996]).

### 3.4.4 Step 4: Data partitioning

Common practice is to divide the time series into three distinct sets called the training, testing and validation[6] (out-of-sample) sets. The training set is the largest set and is used by the neural network to learn the patterns present in the data. The testing set, ranging in size from 10 % to 30 % of the training set, is used to evaluate the generalization ability of a supposedly trained network. The researcher would select the network(s) which perform best on the testing set. A final check on the validation set chosen must strike a balance between obtaining a sufficient sample size to evaluate a trained network and having enough remaining observations for both training and testing. The validation set should consist of the most recent contiguous observations. Care must be taken not to use the validation set as a testing set by repeatedly performing a series of train-test-validation steps and adjusting the input variables based on the network's performance on the validation set (Kaastra and Boyd [1996]).

The testing set can be either randomly selected from the training set or consist

[6]In some of studies (e.g. in the one presented in section 3.3.1), the term *testing set* is used as a name for the validation set.

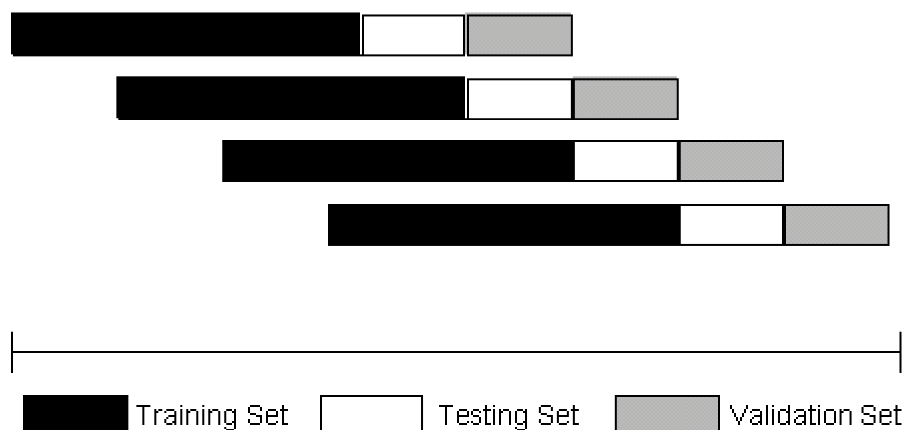



Figure 3.7: A walk-forward sliding windows testing routine (Kaastra and Boyd [1996]).

of a set of observations immediately following the training set. The advantage of randomly selecting testing facts is that the danger of using a testing set characterized by one type of market is largely avoided. For example, a small testing set may only consist of prices in a strong uptrend. The testing set will favour networks which specialize on strong uptrends at the expense of networks which generalize by performing well on both uptrends and downtrends. The advantage of using the observations following the training set as testing facts is that these are the most recent observations (excluding the validation set) which may be more important than older data (Kaastra and Boyd [1996]).

The randomly selected testing facts should not be replaced in the training set because this would bias the ability to evaluate generalization especially if the testing set is large relative to the training set (e.g. 30 %). A deterministic method, such as selecting every $n$th observation as a testing fact, is also not recommended since it can result in cycles in the sampled data due solely to the sampling technique employed (Masters [1993]; Kaastra and Boyd [1996]).

A more rigorous approach in evaluating neural networks is to use a *walk-forward testing* routine also known as either *sliding* or *moving window* testing. Popular in evaluating commodity trading systems, walk-forward testing involves dividing the data into a series of overlapping training-testing-validation sets. Each

set is moved forward through the time series as shown in figure 3.7. Walk-forward testing attempts to simulate real-life trading and tests the robustness of the model through its frequent reatraining on a large out-of-sample data set. In walk-forward testing, the size of the validation set drives the retraining frequency of the neural network. Frequent retraining is more time consuming, but allows the network to adapt more quickly to changing market conditions. The consistency or variation of the results in the out-of-sample sets is an important performance measure (Kaastra and Boyd [1996]).

### 3.4.5 Step 5: Neural network design

There are an infinite number of ways to construct a neural network. *Neurodynamics* and *architecture* are two terms used to describe the way in which a neural network is organized. The combination of neurodynamics and architecture define the neural networks's *paradigm*. Neurodynamics describe the properties of an individual neuron such as its transfer function and how the inputs are combined (Nelson and Illingworth [1991]). A neural network's architecture defines its structures including the number of neurons in each layer and the number and type of interconnections (Kaastra and Boyd [1996]).

The number of input neurons is one of the easiest parameters to select once the independent variables have been preprocessed because each independent variable is represented by its own input neuron. The tasks of selection of the number of hidden layers, the number of neurons in the hidden layers, the number of output neurons as well as the transfer functions are much more difficult (Kaastra and Boyd [1996]).

#### Number of hidden layers

The hidden layer(s) provide the network with its ability to generalize. In practice, neural networks with one and occasionally two hidden layers are widely used and have performed very well. Increasing the number of hidden layes also increases computation time and the danger of overfitting which leads to poor out-of-sample forecasting performance. Overfitting occurs when a forecasting

model has too few degrees of freedom. In other words, it has relatively few observations in relation to its parameters and therefore it is able to memorize individual points rather than learn the general patterns. In the case of neural networks, the number of weights, which is inexorably linked to the number of hidden layers and neurons, and the size of the training set (number of observations) determine the likelihood of overfitting (Masters [1993], Baum and Haussler [1989]). The greater the number of weights relative to the size of the training set, the greater the ability of the network to memorize idiosyncrasies of individual observations. As a result, generalization for the validation set is lost and the model is of little use in actual forecasting (Kaastra and Boyd [1996]). Therefore, it is recommended that all neural networks should start with preferably one or at most two hidden layers. If a four-layer neural network (i.e. two hidden layers) proves unsatisfactory after having tested multiple hidden neurons using a reasonable number of randomly selected starting weights, then the researcher should modify the input variables a number of times before adding a third hidden layer. Both theory and virtually all empirical work suggest that networks with more than four layers will not improve the results (Kaastra and Boyd [1996]).

**Number of hidden neurons**

Despite its importance, there is no "magic" formula for selecting the optimum number of hidden neurons. Therefore researchers fall back on experimentation. However, some rules of thumb have been advanced. A rough approximation can be obtained by the geometric pyramid rule proposed by Masters (Masters [1993]). For a three-layer network with $n$ input neurons and $m$ output neurons, the hidden layer would have $\sqrt{n \times m}$ neurons. The actual number of hidden neurons can still range from one-half to two times the geometric pyramid rule value depending on the complexity of the problem. Baily and Thompson [1990] suggest that the number of hidden neurons in a three-layer neural network should be 75 % of the number of input neurons. Katz [1992] indicates that an optimal number of hidden neurons will generally be found between one-half

to three times the number of input neurons. Ersoy [1990] proposes doubling the number of hidden neurons until the network's performance on the testing set deteriorates. Klimasauskas [1993] suggests that there should be at least five times as many training facts as weights, which sets an upper limit on the number of input and hidden neurons (Kaastra and Boyd [1996]).

It is important to note that the rules which calculate the number of hidden neurons as a multiple of the number of input neurons implicitly assume that the training set is at least twice as large as the number of weights and preferably four or more times as large. If this is not the case, then these rules of thumb can quickly lead to overfitted models since the number of hidden neurons is directly dependent on the number of input neurons (which in turn determine the number of weights). The solution is to either increase the size of the training set or, if this is not possible, to set an upper limit on the number of input neurons so that the number of weights is at least half of the number of training facts. Selection of input variables becomes even more critical in such small networks since the luxury of the presenting the network with a large number of inputs and having it ignore the irrelevant ones has largely disappeared (Kaastra and Boyd [1996]).

Selecting the best number of hidden neurons involves experimentation. Three methods often used are the fixed, constructive and destructive. In the fixed approach, a group of neural networks with different numbers of hidden neurons are trained and each is evaluated on the testing set using a reasonable number of randomly selected starting weights. The increment in the number of hidden neurons may be one, two or more depending on the computational resources available. Plotting the evaluation criterion (e.g. sum of squared errors) on the testing set as a function of the number of hidden neurons for each neural network generally produces a bowl shaped error graph. The network with the least error found at the bottom of the bowl is selected because it is able to generalize best. This approach is time consuming, but generally works very well (Kaastra and Boyd [1996]).

The constructive and destructive approaches involve changing the number of hidden neurons during training rather than creating separate networks each

with a different number of hidden neurons, as in the fixed approach. Many commercial neural network software packages do not support the addition or removal of hidden neurons during training. The constructive approach involves adding hidden neurons until network performance starts deteriorating. The destructive approach is similar except that hidden neurons are removed during training (Kaastra and Boyd [1996]).
Regardless of the method used to select the range of hidden neurons to be tested, the rule is to always select the network that performs best on the testing set with the least number of hidden neurons. When testing a range of hidden neurons it is important to keep all other parameters constant. Changing any parameter in effect creates a new neural network with a potentially different error surface which would needlessly complicate the selection of the optimum number of hidden neurons (Kaastra and Boyd [1996]).

**Number of output neurons**

Deciding on the number of output neurons is somewhat more straightforward since there are compelling reasons to always use only one output neuron. Neural networks with multiple outputs, especially if these outputs are widely spaced, will produce inferior results as compared to a network with a single output (Masters [1993]). A neural network trains by choosing weights such that the average error over all output neurons is minimized. For example, a neural network attempting to forecast one month ahead and six months ahead cattle futures prices will concentrate most of its effort on reducing the forecast with the largest error which is likely the six month forecast. As a result, a relatively large improvement in the one month forecast will not be made if it increases the absolute error of the six month forecasts by an amount greater than the absolute improvement of the one month forecast. The solution is to have the neural networks specialize by using separate networks for each forecast. Specialization also makes the trial and error design procedure somewhat simpler since each neural network is smaller and fewer parameters need to be changed to fine tune the final model (Kaastra and Boyd [1996]).

### Transfer functions

The majority of current neural network models use the sigmoid transfer function, but others such as the tangens hyperbolicus, arcus tangens and linear transfer functions have also been proposed (Kaastra and Boyd [1996]).
Linear transfer functions are not useful for nonlinear mapping and classification. Levich and Thomas [1993] and Kao and Ma [1992] found that financial markets are nonlinear and have memory suggesting that nonlinear transfer functions are more appropriate. Transfer functions such as the sigmoid are commonly used for time series data because they are nonlinear and continuously differentiable which are desirable properties for network learning (Kaastra and Boyd [1996]). Klimasauskas [1993] states that if the network is to learn average behavior a sigmoid transfer function should be used while if learning involves deviations from the average, the tangens hyperbolicus function works best. In a standard backpropagation network, the input layer neurons typically use linear transfer functions while all other neurons use a sigmoid function (Kaastra and Boyd [1996]).
The raw data is usually scaled between 0 and 1 or $-1$ and $+1$, so it is consistent with the type of transfer function which is being used. Linear and mean/standard deviation scaling (see section 2.2.3) are two of the most common methods used in neural networks. In linear scaling all observations are linearly scaled between the minimum and maximum values according to the following formula:

$$SV = TF_{min} + (TF_{max} - TF_{min}) \times \frac{(D - D_{min})}{(D_{max} - D_{min})} \tag{3.1}$$

where $SV$ is the scaled value, $TTF_{min}$ and $TF_{max}$ are the respective minimum and maximum values of the transfer function, $D$ is the value of observation and $D_{min}$ and $D_{max}$ are the respective minimum and maximum values of all observations (Kaastra and Boyd [1996]).

Simple linear scaling is susceptible to outliers because it does not change the uniformity of the distribution, but merely scales it into the appropriate range for the transfer function. In the S&P 500 data presented in figure 3.8, linear

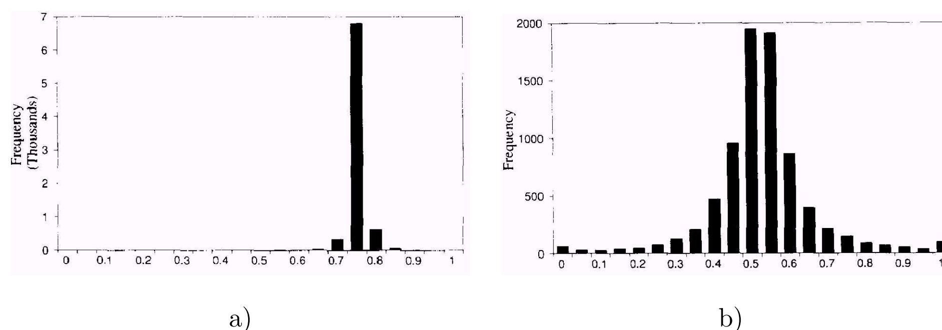



Figure 3.8: Distribution of the scaled values of the S&P 500 closing prices (1962-1993) time series. In figure a) the time series was scaled by means of minimum and maximum scaling, in figure b) the same time series was scaled with mean and standard deviation scaling (Kaastra and Boyd [1996]).

scaling results in 98.6 % of the training facts being contained within 10 % of the neuron's activation range (figure 3.8, a)). Proper training is unlikely to take place with such a distribution. In mean and standard deviation scaling all values plus or minus $x$ number of standard deviations from the mean are mapped to one and zero respectively. All other values are linearly mapped between zero and one. This type of scaling creates a more uniform distribution (figure 3.8, b)) and is more appropriate for data which has not been sampled in any way. Most neural network software programs will automatically scale all the variables into the appropriate range. However, it is always a good idea to look at histograms of the scaled input and output variables (Kaastra and Boyd [1996]).

### 3.4.6 Step 6: Evaluation of the system

The most common error function minimized in neural networks is the sum of squared errors. Other error functions offered by software vendors include least absolute deviations, least fourth powers, asymmetric least squares and percentage differences. These error functions may not be the final evaluation criteria since other common forecasting evaluation methods such as the *mean absolute percentage error* (MAPE) are typically not minimized in neural networks (Kaastra and Boyd [1996]).

In the case of commodity trading systems, the neural network forecasts would be converted into buy/sell signals according to a predetermined criterion. For example, all forecasts greater than 0.8 or 0.9 can be considered buy signals and all forecasts less than 0.2 or 0.1 as sell signals (Hamm et al. [1993]). The buy/sell signals are then fed into a program to calculate some type of risk adjusted return and the networks with the best risk adjusted return (not the lowest testing set error) would be selected. Low forecast errors and trading profits are not necessarily synonymous since a single large trade forecasted incorrectly by the neural network could have accounted for most of the trading system's profits (Kaastra and Boyd [1996]).

### 3.4.7 Step 7: Training the ANN

Training a neural network to learn patterns in the data involves iteratively presenting it with examples of the correct known answers. The objective of training is to find the set of weights between the neurons that determine the global minimum of the error function. Unless the model is overfitted, this set of weights should provide good generalization. The backpropagation network uses a gradient descent training algorithm which adjusts the weights to move down the steepest slope of the error surface. Finding the global minimum is not guaranteed since the error surface can include many local minima in which the algorithm can become "stuck". A momentum term[7] and five to ten random sets of starting weights can improve the chances of reaching a global minimum. This section will discuss when to stop training a neural network and the selection of learning rate and momentum values (Kaastra and Boyd [1996]).

#### Number of training iterations

There are two schools of thought regarding the point at which training should be stopped. The first stresses the danger of getting trapped in a local minimum and the difficulty of reaching a global minimum. The researcher should only stop training until there is no improvement in the error function based on a

[7] For definition of momentum term refer to glossary.

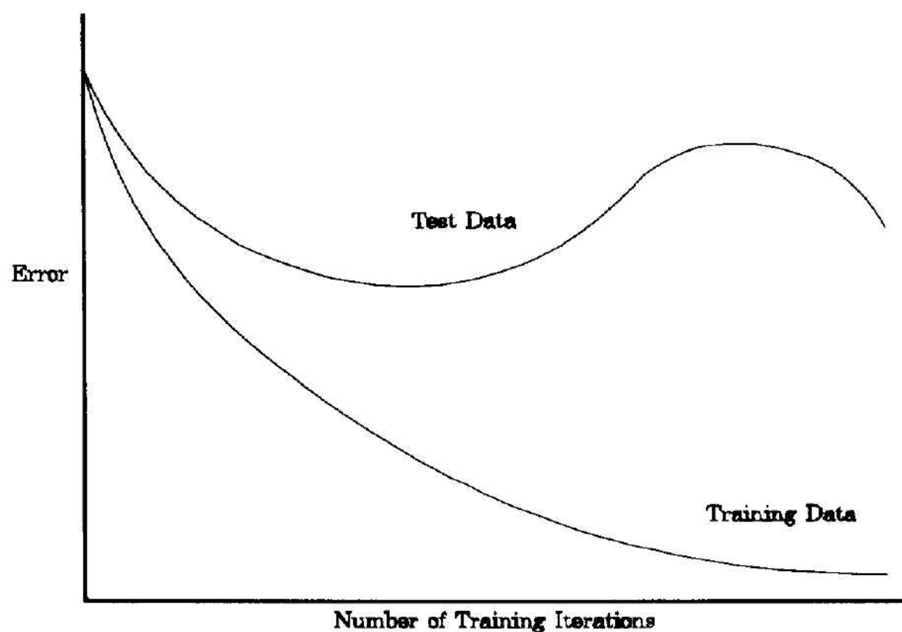



Figure 3.9: Possible neural network training and testing set errors (Kaastra and Boyd [1996]).

reasonable number of randomly selected starting weights (Masters [1993]). The point at which the network does not improved is called convergence. The second view advocates a series of train-test interruptions (Deboeck [1994], Mendelsohn [1993]). Training is stopped after a predetermined number of iterations and the network's ability to generalize on the testing set is evaluated and training is resumed. Generalization is the idea that a model based on a sample of the data is suitable for forecasting the general population. The network for which the testing set error bottoms out is chosen since it is assumed to generalize best (Kaastra and Boyd [1996]).

The criticism of the train-test procedure is that additional train-test interruptions could cause the error on the testing set to fall further before rising again or it could even fall asymptotically (figure 3.9). In other words, the researcher has no way of knowing if additional training could improve the generalization ability of the network especially since starting weights are randomized (Kaastra and Boyd [1996]).

Both schools of thought agree that generalization on the validation set is the ultimate goal and both use testing sets to evaluate a large number of networks. The point at which the two approaches depart centres on the notion of overtraining versus overfitting. The convergence approach states that there is no such thing as overtraining, only overfitting. Overfitting is simply a symptom of a network that has too many weights. The solution is to reduce the number of hidden neurons (or hidden layers if there is more than one) and/or increase the size of the training set. The train-test approach attempts to guard against overfitting by stopping training based on the ability of the network to generalize (Kaastra and Boyd [1996]).

The advantage of the convergence approach is that one can be more confident that the global minimum was reached. Replication is likely more difficult for the train-test approach given that starting weights are usually randomized and the mean correlation can fluctuate wildly as training proceeds. Another advantage is that the researcher has two less parameters to worry about; namely the point at which to stop training and the method to evaluate which of the train-test networks is optimal. An advantage of the train-test approach may be that networks with few degrees of freedom (weights) can be implemented with better generalization than convergence training which would result in overfitting. However, empirical work has not specifically addressed this issue. The train-test approach also requires less training time (Kaastra and Boyd [1996]).

The objective of convergence training is to reach a global minimum. This requires training for a sufficient number of iterations using a reasonable number of randomly selected starting weights. Even then there is no guarantee with a backpropagation network that a global minimum is reached, since it may become trapped in a local minimum. In practice, computational resources are limited and tradeoffs arise. The researcher must juggle the number of input variable combinations to be trained, the interval of hidden neurons over which each network is to be tested, the number of randomly selected starting weights, and the maximum number of runs. For example, 50 input variable combinations tested over three different hidden neurons with five sets of randomly selected

weights, and the maximum number of runs of 4000 result in 3000000 iterations (epochs). The same computational time is required for ten input variable combinations tested over six hidden neurons with ten sets of randomly selected starting weights and 5000 epochs (Kaastra and Boyd [1996]).

One method to determine a reasonable value for the maximum number of runs is to plot the mean correlation, sum of squared errors, or other appropriate error measure for each iteration or at predetermined intervals up to the point where improvement is negligible (usually up to a maximum of 10000 iterations). Each iteration can be easily plotted if the neural network software creates a statistics file or, if this is not the case, the mean correlation can be recorded at intervals of 100 or 200 from the computer monitor. After plotting the mean correlation for a number of randomly selected starting weights, the researcher can choose the maximum number of runs based on the point where the mean correlation stops increasing quickly and flattens (Kaastra and Boyd [1996]).

Many studies that mention the number of training iterations report convergence from 85 to 5000 iterations (Deboeck and Cader [1994], Klaussen and Uhrig [1994]). However, the range is very wide as 50000 and 191400 iterations (Klimasauskas [1993], Odom and Sharda [1992]) and training times of 60 hours have also been reported (Hamm et al. [1993]). Training is affected by many parameters such as the choice of learning rate and momentum values, proprietary improvements to the backpropagation algorithm, among others, which differ between studies and so it is difficult to determine a general value for the maximum number of runs. Also, the numerical precision of the neural network software can affect training because the slope of the error derivative can become very small causing some neural network programs to move in the wrong direction due to round off errors which can quickly build up in the highly iterative training algorithm. It is recommended that researchers determine the number of iterations required to achieve negligible improvement for their particular problem and test as many randomly selected starting weights as computational constraints allow (Kaastra and Boyd [1996]).

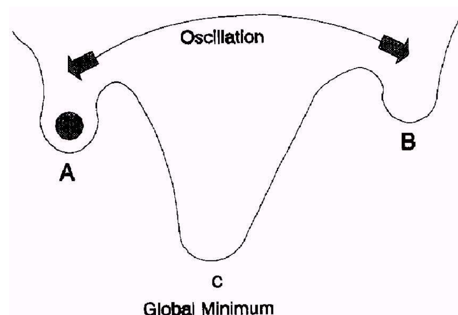



Figure 3.10: Simplified graphical representation of a neural network error surface (Kaastra and Boyd [1996]).

**Learning rate and momentum**

The notion of the learning rate in backpropagation networks has already been introduced in section 2.1.6. As an analogy to the backpropagation training algorithm, one can consider the problem of trying to throw a ball from point A to point C in figure 3.10, although in reality the error surface is multidimensional and cannot be represented in a graphical format. The force on the ball is analogous to the learning rate. Applying too much force will cause the ball to overshoot its target and it may never return to point A or it can oscillate between points A and B. During training, a learning rate that is too high is revealed when the error function is changing wildly without showing a continued improvement. Too little force on the ball and it will be unable to escape from point A which is evident during training when there is very little or no improvement in the error function. A very small learning rate also requires more training time. In either case, the researcher must adjust the learning rate during training or "brainwash" the network by randomizing all weights and changing the learning rate for the new run through the training set (Kaastra and Boyd [1996]).

One method to increase the learning rate and thereby speed up training time without leading to oscillation is to include a momentum term in the backpropagation learning rule. The momentum term determines how past weight changes

affect current weight changes. The modified backpropagation training rule is defined as follows:

$$\Delta w_{ij}(n) = \eta \delta_{ji} x_{ji} + \alpha \Delta w_{ji}(n-1) \tag{3.2}$$

where $\eta$ is the learning rate, $\alpha$ is the momentum term, $\Delta w_{ij}(n)$ is the change of weight $w_{ij}$ at learning epoch $n$ and $x_{ji}$ is the $i$th input to neuron $j$ (Kaastra and Boyd [1996]).

The momentum term suppresses side to side oscillations by filtering out high-frequency variations. Each new search direction is a weighted sum of the current and the previous gradients. Such a two-period moving average of gradients filters out rapid fluctuations in the learning rate. Momentum values that are too great will prevent the algorithm from following the twists and turns in weight space. McClelland et al. [1986] indicate that the momentum term is especially useful in error spaces containing long ravines that are characterized by steep, high walls and a gently sloping floor. Without a momentum term, a very small learning rate would be required to move down the floor of the ravine which would require excessive training time. By dampening the oscillations between ravine walls, the momentum term can allow a higher learning rate to be used (Kaastra and Boyd [1996]).

Most neural network software programs provide default values for learning rate and momentum that typically work well. Initial learning rates used in previous work vary widely from 0.1 to 0.9 Common practice is to start training with a higher learning rate such as 0.7 and decrease as training proceeds. Many neural network programs will automatically decrease the learning rate and increase momentum as convergence is reached (Kaastra and Boyd [1996]).

### 3.4.8 Step 8: Implementation

The implementation step is listed as the last one, but in fact requires careful consideration prior to collecting data. Data availability, evaluation criteria and training times are all shaped by the environment in which the neural network will be deployed. Most neural network software vendors provide the means by

which trained networks can be implemented either in the neural network program itself or as an executable file (Kaastra and Boyd [1996]).

If not, a trained network can be easily created in a spreadsheet by knowing its architecture, transfer functions and weights. Care should be taken that all data transformations, scaling and other parameters remain the same from testing to actual use (Kaastra and Boyd [1996]).

An advantage of neural networks is their ability to adapt to changing market conditions through periodic retraining. Once deployed, a neural network's performance will degrade over time unless retraining takes place. However, even with periodic retraining, there is no guarantee that network performance can be maintained as the independent variables selected may have become less important (Kaastra and Boyd [1996]).

It is recommended that the frequency of retraining for the deployed network should be the same as used during testing on the final model. However, when testing a large number of networks to obtain the final model, less frequent retraining is acceptable in order to keep training times reasonable. A good model should be robust with respect to retraining frequency and will usually improve as retraining takes place more often (Kaastra and Boyd [1996]).

# Chapter 4

# Related Issues

In this chapter personal opinions of the author to different areas of artificial intelligence in general and the specific domain of financial time series prediction with ANNs are presented.

## 4.1 Impact of artificial intelligence to the practice of stock trading

I believe that neural networks and AI in general is changing the practice of stock trading crucially. Since in almost every scientific report or article, results of the applied algorithms are compared to other techniques, it can be assumed that traders will be (or are already) using several algorithms and use the currently best performing one.

My view is supported by the fact that in past, scientific research did not remain unnoticed by financial trading practitioners. So, for example, Fama mentions two points where the research of the Efficient Market Hypothesis (see p. 16) influenced the way of thinking (thus way of acting) of traders ([Fama, 1991, p. 35]):

> Since we are reviewing studies of performance evaluation, it is well to point out here that the efficient-markets literature is a premier

> case where academic research has affected real-world practice. Before the work on efficiency, the presumption was that private information is plentiful among investment managers. The efficiency research put forth the challenge that private information is rare. One result is the rise of passive investment strategies that simply buy and hold diversified portfolios (e.g. the many S&P 500 funds). Professional managers who follow passive strategies (and charge low fees) were unheard of in 1960; they are now an important part of the investment-management industry.

The second change of way of trading follows few lines later ([Fama, 1991, p. 35]):

> The market-efficiency literature also produced a demand for performance evaluation. In 1960, investment managers were free to rest on their claims about performance. Now, performance measurement relative to passive benchmarks is the rule, and there are firms that specialize in evaluating professional managers.

Hence it can very well be assumed that future (or particularly smart today's trader) will use a big arsenal of predictors, but trust only the one which performs best at the moment.

## 4.2 Integrated time series predictor

### 4.2.1 Introduction

In this section an approach to financial time series prediction will be introduced theoretically, which appears to be most promising according to my opinion. The imaginary predictor presented here is regarded as the logical "next step" in the domain of financial time series prediction research.

### 4.2.2 Motivation

> *Nature uses only the longest threads to weave her patterns, so that each small piece of her fabric reveals the organization of the entire tapestry.*
>
> Richard Feynman

Financial time series aren't natural phenomena, they are mere images of natural phenomenon of economy[1], like electromagnetic rays and star dust are distant appearance of stars observed by telescopes or spacecraft. Financial time series can not be separated from the process that generates them.

Financial time series can be compared with time series in other scientific fields, e.g. the electrocardiogram (ECG) in medicine. It provides valuable information about the health of a person, but nobody assumes that a precise diagnosis can be made using *only* the ECG or statistical data derived from it. In this case, the underlying process is the activity of human organism as a whole and the ECG an image of one part (out of many parts) of its activity. In case of financial time series, they are the images and the underlying process is the economy.

Paraphrasing the quote at the beginning of this section, the financial time series are the threads of a large tapestry, the economy.

Contrary to this intuitive idea, financial prediction techniques discussed in this work, try to separate financial time series from the underlying process and to predict them purely on statistical basis. As we saw above, this works very well. But why are neural networks claimed to be good predictors? To author's opinion, because there is nothing better currently present.

A sensibly designed prediction algorithm should take into account not only statistical information, but also the most important current properties of the underlying process. Such a predictor is called by the author an *integrated time series predictor* , since it integrates several different subsystems.

[1] The author claims that stock and option trading are part of the economy as a whole which is a natural phenomenon. It is natural, because every human society has to decide "what, how and for whom to produce" (study of these three w-questions is called "economics", [Begg et al., 1997, p. 2]). Hence, economical system is present everywhere in one form or another.

### 4.2.3 Possible Implementations

What components should an integrated time series predictor consist of? In order to make this decision, we will analyze the processes that a human trader executes when making prediction about the future.

Firstly, a human trader is able to process newspaper articles, news related to the company or industry of interest and other external information which in most cases is available in form of text.

Secondly, a human trader has a profound economic education, which allows him to extract important causal connections (patterns) in the information he receives.

Thirdly, a human trader has a long experience as trader. This means that through long experience he learned certain general rules ("rules of thumb") using which he manages to make decision even with incomplete or false information at hand. Normally, these decisions are correct. "Subconscious" intuitive knowledge (affinity towards taking a certain - in most cases correct - action without the ability to explain the reason for doing so verbally) also falls in this category.

Fourthly, a human trader has powerful statistics software at hand, which allow him to develop an understanding of current market situation from the statistician's point of view (the point of view of financial time series prediction).

What happens, if such a trader makes a mistake? Since human species usually work in teams and the work of one person is often verified by other team members, it can be assumed that most random mistakes are ruled out provided that the team as a whole works efficiently. This is the fifth feature of a human trader.

The proposed integrated time series predictor mimics most of these features.

The structure of an integrated time series predictor that possesses many of the above properties is presented in figure 4.1. The integrated time series predictor consists of three subsystems. The *linguistic subsystem* has the task of extracting relevant information from textual data. The *text reader* is a simple "non-AI" system that simply gathers textual data which are relevant for period of time and application domain in question. The text classification subsystem serves the purpose of deriving simple information out of textual data by means

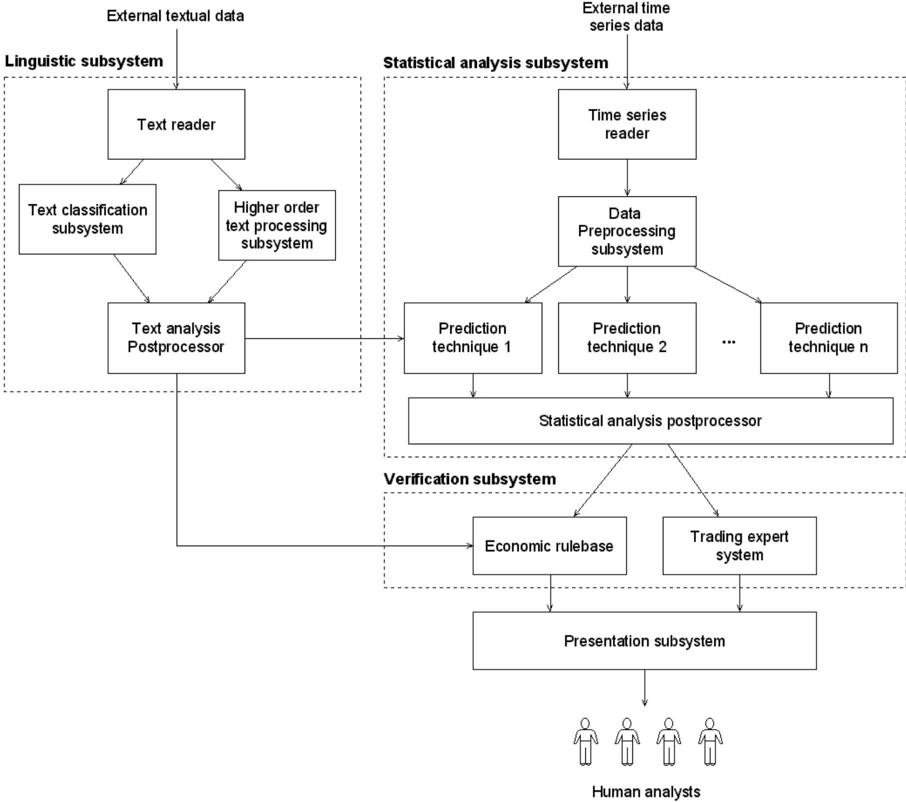



Figure 4.1: Logical structure of an integrated time series predictor

of text classification. That means, that it has been trained to recognize texts that precede large movements of the quantity of interest (e.g. the stock price). This subsystem works mainly on statistical basis and is only able to give basic (simple) information, such as whether the stock index will rise or fall on the next day. There exist a lot of algorithms for text classification, many of them being implemented as part of the famous *Rainbow* package (McCallum [1998]).
The *higher order text processing subsystem* is a more sophisticated text analyzer. It should be able to extract more complex relationships from the available text data, perhaps using some natural language processing (NLP) technique. Such an NLP mechanism would be designed specifically for the domain of financial texts, i.e. it is not necessary for this subsystem to be a universal text analyzer. The higher order text processing subsystem could be implemented as a text parser connected with some semantic base and a comprehensive sense derivation mechanism. Such a system could process huge amounts of information (current state of economy, large companies' news, macroeconomic events etc) contained in textual form (e.g. the articles of financial newspapers, newsgroups, WWW texts).

The *text analysis postprocessor* is a "non-AI" mechanism for formatting the results of textual analysis into a form which can be used by the statistical techniques and in scope of the verification of them.

The *statistical analysis subsystem* is the module which forms predictions on the basis of statistical analysis of time series. The *data preprocessing subsystem* incorporates the vast variety of preprocessing methods already described in section 2.2.3. The most important property of the statistical module is the presence of many different prediction techniques. There are many possibilities how they can used for prediction of further action:

**Voting** From all possible prediction alternatives the one is chosen, for which the majority of the techniques "vote",i.e. if there are three prediction alternatives "tomorrow the stock price will fall" (1), "tomorrow the stock price will rise" (2) and "tomorrow there will be insignificant changes in the stock price" (3), and 3 prediction techniques predict "1", 5 predict "2"

and and one predicts "3", then prediction "2" is taken as final prediction since the majority of techniques "voted" for it.

**Best performing technique wins** The other approach may be the same as described in Tino et al. [2000], namely that all algorithms are run on past data (e.g. the last 100 days) and then the best performing one is chosen as a predictor for the current day.

The necessity for this mechanism arises from time-varying nature of financial time series already described in section 1.1. The *statistical analysis preprocessor* is a mechanism that summarizes the prediction data of the various techniques and calculates other important information. Such information may be a measure of confidence in the prediction made by a particular technique (e.g. such as those proposed in Dybowski and Roberts [2001]). Other useful postprocessing step would be the extraction of symbolic representations out of neural networks. Further, the statistical analysis postprocessor serves the purpose of formulating a reaction to the prediction, being in most cases a certain trading action (buy, sell).
These reactions are being verified by two databases, which represent general knowledge about

1. how economy functions (*economic rulebase*)
2. how successful traders act in similar situations (*trading expert system*).

They have the task of verifying that the trading actions proposed by the statistical analysis subsystem are consistent with economic "common sense" and experience of successful traders.

### 4.2.4 Justification of the integrated time series predictor

Apart from the linguistic subsystem, all components of the integrated time series predictor presented above are well-known (and practised) way of time series prediction. As already mentioned in comments to the examples (see section 3.3), application of different prediction techniques is necessary due to the time-variability of financial time series and is done in most studies of ANNs in finance.

The verification subsystem may sound as a novelty, but it is not. Verification of statistical time series prediction is currently done by other machines, namely humans. Transferring their knowledge to databases may be an engineering, but in no case a scientific innovation.

Hence, the only point that needs a rational explanation is the presence of the linguistic subsystem. Apart from the intuition that the textual information contained in various sources (newspapers, journals, newsgroups, WWW page) is useful for making good financial predictions, two empirical studies on incorporation of textual information into financial time series prediction will be briefly presented. These studies provide evidence of usefulness of incorporation of textual data into prediction algorithms.

In the study described in Thomas and Sycara [2000], the textual information comes from the postings on a web bulletin board. These data are processed in two ways:

1. text classification that establishes a relationship between the up/down price movements and the statistical properties of the bulletin board messages of a certain day, i.e. the entire bulletin board corpus is used to form the "up" or "down" prediction

2. genetic algorithm that learns simple rules, which are based on numeric properties of the textual data set (such as number of messages posted on a day, the number of words etc).

The integration of the two approaches resulted in a significant improvement of performance in terms of profit.

The architecture of the second predictor (described in Wüthrich et al. [1998]) that uses textual data is given in figure 4.2. *Old news* and *Old index values* contain the training data, the news and closing values of the last one hundred stock trading days (Wüthrich et al. [1998]). *Keyword records* contains over four hundred individual sequences of words such as "bond strong", "dollar falter", "property weak", "dow rebound", "technology rebound strongly" etc (Wüthrich et al. [1998]). These are sequences of words provided once by a domain expert

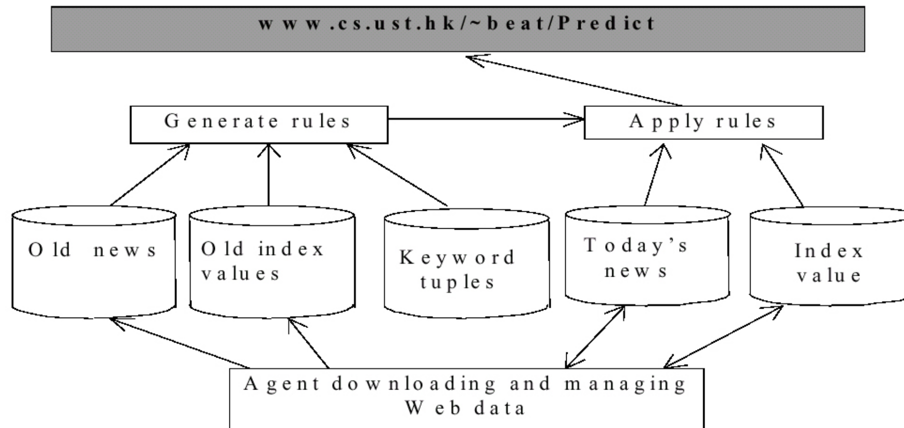



Figure 4.2: Architecture of textual financial time series predictor described in Wüthrich et al. [1998].

and judged to be influential factors potentially moving stock markets (Wüthrich et al. [1998]).

Given the current data, the prediction is done as follows (Wüthrich et al. [1998]):

1. The number of occurrences of the keyword records in the news of each day is counted.

2. The occurrences of the keywords are then transformed into weights. This ways, for each day, each keyword gets a weight.

3. From the weights and the closing values of the training data, probabilistic rules are generated.

4. The generated rules are applied to today's news. This predicts whether a particular index such as the Dow will go up, moves down or remains steady.

5. From the prediction whether the Dow goes up, down or remains steady, and from the latest closing value also the expected actual closing value such as 8393 is predicted.

6. The generated predictions are then moved to the Web page `http://www.cs.ust.hk/~beat/Predict` where each day at 7:45 local time in Hong Kong the daily stock market forecasts can be followed.

The average accuracy reported by Wüthrich et al. [1998] is 43.6 % correct, 37.4 % "slightly wrong" (the system predicts up or down and it was actually steady; or, the system predicts steady and it was actually up or down) and 19 % wrong (the system expects the index to go up and it moves down, or vice versa).
However, one should not forget, that prediction accuracy is not always proportional to the profitability of a prediction algorithm (as we saw in section 3.3.1).

## 4.3 The perfection aspiration

The study of several AI surveys like Johnson-Laird [1996] suggests that scientists involved in AI research seem to rule out any possibility of emergence of intelligent machines due to unrealistic idea of how complex systems are developed.
I do not agree with such way of thinking and intend to show its weaknesses on the following lines.
The thread leading through this explanation is natural language processing, but the problem discussed below affects not only NLP, but AI in general.

### 4.3.1 Problem definition

The problem I am discussing in this section is a problem of wrong thinking. After surveying the capabilities of today's computers and current stage of development in computer science and other scientific disciplines, one wonders why the AI has such a little impact on our everyday lives.
For example, there exists a scientific basis for development of natural language interfaces[2] for software programs, yet only few of them have such NLP interfaces. At the same time, often the computer programs are quite complicated so that

[2] "Natural language interfaces" means text-based interface, not speech recognition

any help is highly appreciated by the user, even if the answers to his questions are not always precise and even if the computer interacts with the user in a simple English.

Often, scientists claim that the reason for this lies in inferior capabilities of today's software or restricted computational power. While the latter argument will exist probably always, the former deserves more attention.

It is well known that computers can not do many things (natural language processing among them) with the same ease as humans. They are imperfect language processors, image analysts and thinkers. But does this mean, that they cannot be useful unless they are perfect?

To the author's opinion, they *can* be useful even if they aren't perfect. Consider for example the domain of automatic translation. Almost everyone knows jokes about early experiments in this domain. Even today, the computers are not capable to make precise translations automatically. But they *can* and do assist human translators through making "raw" translations which then are verified by a human translator. Automatic translators are far from being perfect, yet they are useful by making the process of translation more efficient.

Brought to a point, the way of thinking of many scientists can be expressed as: wait until we are able to build perfectly intelligent (human-like) machine, and then it will be possible to make any use of AI. This way of thinking is typical particularly in the area of NLP.

### 4.3.2 Counterargument 1: Artificial versus Natural Intelligence

The first point I criticize on this way of thinking is the mixing up artificial and natural intelligence. Artificial (computer driven) and natural (powered by the natural computers in human brain) are different *both* in function and in purpose. Functional difference is given by the fact that artificial computers and natural computers are machines, which are implemented using completely different el-

ements at the low level (flip-flops on one side, neurons on another[3]). However, it may be the case that both of them are similar on higher levels (e.g. the deduction and induction procedures, learning).

Difference in purpose stems from the fact that natural (human) computer is designed to steer its possessor in his environment ensuring that he can adapt to, survive and reproduce himself in it. Contrary to natural intelligence mechanisms, AI exists to solve exactly those problems, which humans can not. Hence, it may be very difficult and perhaps not even sensible to try to re-invent all human abilities in machines.

Since natural and artificial intelligence differ in function it cannot be guaranteed that all human abilities can be implemented in machines; and since natural and artificial intelligence differ in purpose it cannot be guaranteed that such an implementation will be of any use.

### 4.3.3 Counterargument 2: The Engineering Approach to Development of Complex Systems

Another argument against the perfection aspiration is the way great breakthroughs in other scientific disciplines occurred.

Consider, for example, the space flight. Making his early experiments, Ziolkovsky[4] and other scientists of his time did not attempt to build a space station. Instead, the development of spacecraft proceeded in many stages:

1. Ballistic rocket
2. Transcontinental rocket
3. Artificial Earth's satellite
4. Safe flight of a dog into (and return from) space

[3]Since the human brain is not researched in enough detail, "neurons" should be interpreted as "most basic element of human computational machinery".

[4]Constantin Eduardovich Ziolkovsky (1857-1935) was a Russian scientist who in 1903 laid out fundamental principles of rocketry and thus formed the theoretical basis for space flight. His researches were essential for the exploration of space, on the Soviet and Russian as well as on the western side (P.Hürzeler [2001]).

5. Safe flight of a human into (and return from) space
6. operating a space station on earth's orbit where humans can safely live for long periods of time.

Each individual stage was useful in some way. On each stage, there was always a some advantage compared to previous stages (e.g. ability to transport nuclear bombs over longer distances, more sophisticated experiments etc).

The same approach was taken not only by Korolyov[5] and other space researchers, but also by the best engineer of all times: the evolution. Beginning with one cell organisms, the development proceeded to more complex organisms and ended with humans. It did not attempt to proceed from the level of amoebas directly to a homo sapiens sapiens.

In requiring perfection from computers, the scientists who adopt the way of thinking criticized here, demand from the engineer to create a homo sapiens at a stage where not all dinosaurs are dead.

The "right", more logical (and more probable) way the development of AI will take, can be compared with the development of the methods which are subject of this work: financial time series predictors. Beginning with simple ARIMA models, the development proceeded to more powerful GARCH models and to neural networks, and is currently at the stage where statistical time series processing and textual data processing are being integrated in order to achieve a further, small but tangible improvement. Note that at each stage of development, there always was an improvement associated with a new technique.

### 4.3.4 Counterargument 3: What for?

Third counterargument of the perfection aspiration is the highly questionable issue of whether a perfectly human-like machine will be of any use. In the history of mankind, it happened several times that superior scientific achievements were convicted to death in archives without any chance to be used broadly.

[5]Sergey Pavlovich Korolyov (1907 - 1966) "chief designing engineer" of the Soviet space programme, who leaded several highly successful space projects (e.g. the first satellite) which often outperformed analogous western ones.

Our grandparents witnessed the advent of atomic bombs, being even nowadays the most efficient weapon available. Despite its advantages as a weapon and - most important - as means of diplomatic pressure[6], it is not widely used neither the one way nor the other[7].

We witness the advent of another highly promising technology, the science fiction authors of past have dreamed of - the genetic engineering (in particular, cloning). Undoubtedly being a scientific breakthrough, its fruits will hardly be reaped to full extent. Due to high damage potential in case of misuse, some of the leading researchers discontinued their research towards cloning of humans. It can be anticipated that many branches of genetic engineering will be prohibited on an international basis.

Therefore, one can assume that an intelligent, human-like machine will not be used broadly, if developed. Even today, AI is often associated with fear that computers may become too intelligent and make important decisions by themselves, without to consult the humans[8]. Hence, one can assume that long before any human-like computer is developed, the research of it will be either prohibited or restricted due to public protests.

Perhaps we will never have robots in our homes being as natural as humans but always having a shut-down option...

## 4.4 Problems of ANNs in finance

Undoubtedly being much more powerful than traditional time series processing techniques, neural networks have several disadvantages, among which Zekic [1998] mentions

[6]After all, the nuclear weapons of cold war brought the world 46 years of relative (compared to present day) peace.

[7]To author's opinion, many conflicts could be solved if their participants would face the threat of nuclear assault in case they reject to negotiate with each other.

[8]To author's opinion the danger of computer system stems not from their intelligence, but from their complexity and a more difficult testability. A wrongly programmed airplane on-board computer, a cruise missile with defective battery and a buggy spacecraft software resulted in damage because of its complexity, not intelligence.

1. NNs require very large number of previous cases
2. "the best" network architecture (topology) is still unknown
3. for more complicated networks, reliability of results may decrease
4. statistical relevance of the results is needed
5. a more careful data design is needed

Most of these issues have already been adressed in previous sections (particularly section 3.4). The fourth problem identified by Zekic [1998] is a severe one, but the possibilities of evaluating statistical significance of the results produced by neural networks are being researched, e.g. in Dybowski and Roberts [2001], who propose a method of evaluating confidence intervals of the results generated by neural network.

The problem of determination of the optimal topology (point 2) is also being researched. For example, the so-called TACOMA neural network is able to determine the optimal topology automatically (without user intervention) and "grows" minimal neural networks that approximate the target function. The network created in this way is then trained using a variant of the backpropagation algorithm (for description of the TACOMA model see Lange et al. [1994]). In theory, a combination of the SCG learning algorithm (discussed in section 2.1.7) and TACOMA would give rise to a ANN system that does not need any settings to be done by the user at all because the optimal topology is discovered by the TACOMA algorithm and the SCG learning algorithm has no parameters to be set by the user[9].

Further problem of neural networks in finance not mentioned by Zekic [1998], is the slow transfer of theoretical knowledge into practice of ANN application. In particular, this applies to the learning algorithms used. Despite the presence of the highly efficient (in terms of learning speed) SCG algorithm for almost 10 years now, 95 % of neural network systems examined by Wong et al. [1997b]

[9]Author's attempts to empirically research the possible advantages of a TACOMA/SCG combination failed due to lack of a currently working and available TACOMA implementation (Zell [2002]).

employ the older backpropagation algorithm with all its disadvantages.

Last but not least, there is also a problem of interpretability of the results produced by neural networks. Contrary to traditional time series processing methods that have relatively few parameters, trained neural networks are hard to interpret by humans due to the large number of parameters (weights). The incomprehensibility of trained neural networks is a limiting factor not only for the application of neural networks to the particular field of financial time series prediction, but a general problem of neural networks. The author's opinion on this topic is supported by the explanations in Craven and Shavlik [1996] and Duch et al. [2000]. The domain of *symbolic rule extraction* from neural networks appears to be a highly promising research direction.

# Chapter 5

# Conclusions

In scope of the present final year project an investigation of the current stage of development of artificial neural networks in the domain of financial time series prediction has been carried out. Being an important part of the topic of the work, the specific properties of common financial time series were studied (section 1.1, p. 13) and traditional approaches to financial time series prediction have been presented (section 1.4, p. 24) and partially demonstrated on practical examples (appendices A (p. 164), C (p. 169) and D (p. 172)). A detailed treatment of artificial neural networks (chapter 2, p. 35) including the specific topics of data pre- and postprocessing (section 2.2, p. 62) and time series processing (section 2.3, p. 79) is followed by an explanation of issues involved in application of artificial neural networks to financial time series forecasting (chapter 3, p. 89). In particular, a methodology for designing neural network based prediction systems is outlined (section 3.4, p. 102) as well as selected recent research results (section 3.3, 95).

The final part of the work is a discussion of several related issues of the use of neural networks (chapter 4, p. 122). Most promising research directions from the author's point of view are presented concerning both neural network driven time series prediction as well as artificial intelligence in general.

Main findings of the work can be summarized as follows:

- Neural networks are a powerful tool for financial time series prediction, capable of outperforming most other known algorithms (section 3.2, p. 94).

- The potentially superior predictive power of neural networks can be exploited only if specific properties of the time series data are accounted for by means of appropriate pre- and postprocessing mechanisms (section 3.4.3, p. 105).

- Most promising directions of the research in the domain of neural networks in financial time series prediction are incorporation of textual data into prediction systems, statistical evaluation of neural network performance and improving the comprehensibility of neural networks by symbolic rule extraction (chapter 4, p. 122).

# Bibliography

M. Anthony and N. L. Biggs. A computational learning theory view of economic forecasting with neural nets. In A. Refenes, editor, *Neural Networks in the Capital Markets*. John Wiley and Sons, 1995. (reference found in Lawrence et al. [1996]).

V. I. Arnold. On functions of three variables. In *Doklady Akademiia Nauk SSSR*, volume 114 (4), pages 679–681, 1957. (reference found in [Bishop, 1996, p. 137]).

D. Baily and D. M. Thompson. Developing neural network applications. *AI Expert*, pages 33–41, September 1990. (reference found in Kaastra and Boyd [1996]).

E. B. Baum and D. Haussler. What size net gives valid generalization? *Neural Computation*, 6:151–160, 1989. (reference found in Kaastra and Boyd [1996]).

T. Baumann and A. J. Germond. Application of the Kohonen Network to Short-Term Load Forecasting. In *Proc. PSCC*, Avignon, 1993. (reference found in Dorffner [1996]).

D. Begg, S. Fischer, and R. Dornbusch. *Economics*. McGraw-Hill, fifth edition, 1997. ISBN 0 07 709412-3.

A. Beltratti. Answer to the inquiry concerning profitability of algorithms with respect to their availability to other market participants, 2002. (personal communication).

A. Beltratti, S. Margarita, and P. Terna. *Neural Networks for Economic and Financial Modelling*. ITCP, London, 1996.

Y. Bengio. *Neural Networks for Speech and Sequence Recognition*. Thomson, London, 1995. (reference found in Dorffner [1996]).

C. Bishop. *Neural Networks for Pattern recognition*. Oxford University Press, 1996.

P. E. Black, 2002. URL `http://www.nist.gov/dads/HTML/bigOnotation.html`. (URL accessed on February 28, 2002).

B. Borchers. ARMA processes, 2001. URL `http://www.ees.nmt.edu/Geop/Classes/GEOP505/Docs/arimaforcast.pdf,http://www.ees.nmt.edu/Geop/Classes/GEOP505/Docs/arima.pdf,http://www.ees.nmt.edu/Geop/Classes/GEOP505/Docs/arima2.pdf`.

H. Bourlard and N. Morgan. Merging Multilayer Perceptrons and Hidden Markov Models: Some Experiments in Continous Speech Recognition. Technical Report TR-89-33, Int. Computer Science Institute (ICSI), Berkeley, CA, 1989. (reference found in Dorffner [1996]).

J. Y. Campbell, A. W. Lo, and A. C. MacKinlay. *The Econometrics of Financial Markets*. Princeton University Press, 1997.

P. Castiglioni. Glossary of terms used in time series analysis of cardiovascular data, 2001a. URL `http://www.cbi.polimi.it/glossary/Home.html`. (URL accessed on April 25, 2002).

P. Castiglioni. Glossary of terms used in time series analysis of cardiovascular data, 2001b. URL `http://www.cbi.polimi.it/glossary/Home.html`. (URL accessed on April 25, 2002).

K. Chakraborty, K. Mehrota, C. K. Mohan, and S. Ranka. Forecasting the Behavior of Multivariate Time Series Using Neural Networks. *Neural Networks*, 5(6):961–970, 1992. (reference found in Dorffner [1996]).

T. Chenoweth and Z. Obradovic. A multicomponent nonlinear prediction system for the s&p 500 index. *Neurocomputing*, 10(3):275–290, 1996. URL `http://citeseer.nj.nec.com/chenoweth96multicomponent.html`. (URL accessed on March 31, 2002).

M. W. Craven and J. W. Shavlik. Extracting Tree-Structured Representations of Trained Networks. In *Advances in Neural Information Processing Systems*, volume 8. MIT Press, Cambridge, MA, 1996.

A. V. Deardorff. Deardorff's glossary of international economics, 2001. URL `http://www-personal.umich.edu/~alandear/glossary/e.html`. (URL accessed on April 25, 2002).

H. Debar and B. Dorizzi. An Application of a Recurrent Network to an Intrusion Detection System. In *IJCNN International Joint Conference on Neural Networks*, pages 478–483, Baltimore, 1992. IEEE. (reference found in Dorffner [1996]).

G. J. Deboeck, editor. *Trading on the Edge: Neural, Genetic, and Fuzzy Systems for Chaotic Financial Markets*. Wiley, New York, 1994. (reference found in Kaastra and Boyd [1996]).

G. J. Deboeck and M. Cader. Trading U.S. treasury notes with a portfolio of neural net models. In G. J. Deboeck, editor, *Trading on the Edge: Neural, Genetic, and Fuzzy Systems for Chaotic Financial Markets*, pages 102–122. Wiley, New York, 1994. (reference found in Kaastra and Boyd [1996]).

M. W. M. Donders and T. C. F. Vorst. The impact of firm specific news on implied volatilities. *Journal of Banking & Finance*, 20:1447–1461, 1996.

G. Dorffner. Neural networks for time series processing. *Neural Network World*, 6(4):447–468, 1996. URL `http://citeseer.nj.nec.com/179780.html`. (URL accessed on April 26, 2002).

G. Dorffner, E. Leitgeb, and H. Koller. Toward Improving Exercise ECG for Detecting Ischemic Heart Disease with Recurrent and Feedforward Neural

Nets. *Neural Networks for Signal Processing IV, IEEE*, pages 499–508, 1994. (reference found in Dorffner [1996]).

C. Dougherty. *Introduction to Econometrics*. Oxford University Press, New York, 1992.

W. Duch, R. Adamczak, and K. Grabczewski. A new methodology of extraction, optimization and application of crisp and fuzzy logical rules. *IEEE Transactions On Neural Networks*, 11(2), March 2000.

R. Dybowski and S. J. Roberts. Confidence intervals and prediction intervals for feed-forward neural networks. *Clinical Applications of Artificial Neural Networks*, pages 298–326, 2001. URL `http://www.dybowski.com/papers/nnerr.pdf`. (URL accessed on April 1, 2002).

D. Eddelbüttel and T. H. McCurdy. The Impact of News on Foreign Exchange Rates: Evidence from High Frequency Data, May 1998.

O. Ersoy. Tutorial at Hawaii International Conference on Systems Sciences, January 1990. (reference found in Kaastra and Boyd [1996]).

R. C. Fair. Events that shook the market, 2000. URL `http://citeseer.nj.nec.com/fair00events.html`. (URL accessed on March 28, 2002).

R. C. Fair. Shock Effects on Stocks, Bonds, and Exchange Rates, January 2001.

E. F. Fama. The behaviour of stock market prices. *Journal of Business*, pages 34–105, January 1965. (reference found in Lawrence et al. [1996]).

E. F. Fama. Efficient capital markets: A review of theory and empirical work. *Journal of Finance*, pages 383–417, May 1970. (reference found in Lawrence et al. [1996]).

E. F. Fama. Efficient capital markets: Ii. *The Journal of Finance*, 46(5):1575–1617, December 1991.

R. A. Fisher. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7:179–188, 1936. (reference found in Bishop [1996]).

P. E. Gill, W. Murray, and M. H. Wright. *Practical Optimization*. Academic Press Inc., London, 1981. (reference found in Møller [1993]).

P. Glewwe. Time Series Analysis III, 2000. URL `http://www.apec.umn.edu/faculty/pglewwe/Ap821227.pdf`. (URL accessed on October 12, 2001).

A. Gordon, J. P. H. Steele, and K. Rossmiller. Predicting Trajectories Using Recurrent Neural Networks. In C. H. D. et al., editor, *Intelligent Engineering through Artificial Neural Networks*, pages 365–370. ASME Press, New York, 1991. (reference found in Dorffner [1996]).

J. Gregg. Introduction to Taylor series, 2001. URL `http://www.leibnizsoftware.com/about/taylor.pdf`. (URL accessed on November 10, 2001).

R. Gutierrez-Osuna. Dimensionality reduction (lda), 2001. URL `http://www.wright.edu/~ricardo.gutierrez-osuna/courses/cs790_wi02/l6.pdf`. (URL accessed on March 21, 2002).

L. Hamm, B. W. Brorsen, and R. Sharda. Futures trading with a neural network. In *NCR-134 Conference on Applied Commodity Analysis, Price Forecasting and Market Risk Management Proceedings*, pages 286–296, Chicago, 1993. (reference found in Kaastra and Boyd [1996]).

T. Hellström and K. Holmström. Predicting the Stock Market. Technical Report IMa-TOM-1997-07, Center of Mathematical Modeling, Department of Mathematics and Physis, Mälardalen University, Västeras, Sweden, August 1998.

G. Herman. Glossary of linear algebra terms, 1996. URL `http://www.ms.washington.edu/courses/math387-lamp/glossary.html`. (URL accessed on April 4, 2002).

M. R. Hestenes and S. Stiefel. Methods of Conjugate Gradient for Solving Linear Systems. *J. Res. Nat. Bur. Standards*, 49:409–436, 1952. (reference found in Møller [1993]).

D. Hilbert. Mathematische Probleme. In *Nachrichten der Akademie der Wissenschaften Göttingen*, pages 290–329, 1900. (reference found in [Bishop, 1996, p. 137]).

K. Holmström. Technical analysis models for the prediction of financial time series. Technical Report IMa-TOM-1997-10, Department of Mathematics and Physics, Mälardalen University, Sweden, 1997. (reference found in Hellström and Holmström [1998]).

D. R. Hush and J. M. Salas. Improving the learning rate of backpropagation iwth the gradient re-use algorithm. In *IEEE International Conference on Neural Networks*, volume 1, pages 441–447, San Diego, CA, 1988. IEEE. (reference found in [Bishop, 1996, p. 272]).

R. Jain, R. Kasturi, and B. G. Schunck. *Machine Vision*. McGraw-Hill, 1995.

E. M. Johansson, F. U. Dowla, and D. M. Goodman. Backpropagation learning for multi-layer feed-forward neural networks using the conjugate gradient method. *International Journal of Neural Systems*, 2(4):291–301, 1991. (reference found in Møller [1993]).

P. Johnson-Laird. *Der Computer im Kopf ; Formen und Verfahren der Erkenntnis*. Deutscher Taschenbuch Verlag, München, 1996. ISBN 3-423-30499-5.

I. Kaastra and M. Boyd. Designing a neural network for forecasting financial and economic time series. *Neurocomputing*, 10:215–236, 1996. URL `http://www.geocities.com/francorbusetti/KaastraArticle.pdf`. (URL accessed on April 26, 2002).

J. P. Kahane. Sur le theoreme de superposition de Kolmogorov. In *Journal of Approximation Theory*, volume 13, pages 229–234, 1975. (reference found in [Bishop, 1996, p. 137]).

G. W. Kao and C. K. Ma. Memories, heteroscedasticity and prices limit in currency futures markets. *J. Futures Markets*, 12:672–692, 1992. (reference found in Kaastra and Boyd [1996]).

J. O. Katz. Developing neural network forecasters for trading. *Technical Analysis of Stocks and Commodities*, pages 58–70, April 1992. (reference found in Kaastra and Boyd [1996]).

E. Kerr. Financial Management Study Notes ; Capital Asset Pricing Model, 1997. URL `http://www.herts.ac.uk/business/finman/ek422.htm`.

K. L. Klaussen and J. W. Uhrig. Cash soybean price prediction with neural networks. In *NCR-134 Conference on Applied Commodity Analysis, Price Forecasting and Market Risk Management Proceedings*, pages 56–65, Chicago, 1994. (reference found in Kaastra and Boyd [1996]).

C. C. Klimasauskas. Applying neural networks. In R. R. Trippi and E. Turban, editors, *Neural Networks in Finance and Investing: Using Artificial Intelligence to Improve Real World Performance*, pages 64–65. Probus, Chicago, 1993. (reference found in Kaastra and Boyd [1996]).

J. F. Kolen. *Exploring the Computational Capabilities of Recurrent Neural Networks*. Ohio State University, 1994. (reference found in Dorffner [1996]).

A. N. Kolmogorov. On the representation of continous functions of several variables by superposition of continous functions of one variable and addition. In *Doklady Akademiia Nauk SSSR*, volume 114 (5), pages 953–956, 1957. (reference found in [Bishop, 1996, p. 137]).

M. A. Kramer. Nonlinear principal component analysis using autoassociative neural networks. *AIChe Journal*, 37(2):233–243, 1991. (reference found in Bishop [1996]).

J. Lange, H. Voigt, and D. Wolf. Growing artificial neural networks based on correlation measures. In *IEEE International Conference on Neural Networks 1994 as Part of the IEEE World Congress on Computational Intelligence*, volume 2, pages 1355–1358, Orlando, 1994. URL `http://citeseer.nj.nec.com/lange94growing.html`. (URL accessed on February 28, 2002).

S. Lawrence, A. C. Tsoi, and C. L. Giles. Noisy time series prediction using symbolic representation and recurrent neural network grammatical inference. Technical Report UMIACS-TR-96-27 and CS-TR-3625, Institute for Advanced Computer Studies, University of Maryland, College Park, MD 20742, 1996.

R. M. Levich and L. R. Thomas. The significance of technical trading rule profits in the foreign exchange market: A bootstrap approach. In *Strategic Currency Investing - Trading and Hedging in the Foreign Exchange Market*, pages 336–365. Probus, Chicago, 1993. (reference found in Kaastra and Boyd [1996]).

G. G. Lorentz. On the 13th problem of hilbert. In *Proceedings of Symposia in Pure Mathematics*, volume 13, pages 419–429, Providence, RI, 1976. American Mathematical Society. (reference found in [Bishop, 1996, p. 137]).

D. Lowe and A. R. Webb. *Time series prediction by adaptive networks: A dynamical systems perspective.* IEEE Computer Society Press, 1991. (reference found in Hellström and Holmström [1998]).

M. Magdon-Ismail, A. Nicholson, and Y. S. Abu-Mustafa. Financial Markets: Very Noisy Information Processing. *Proceedings of the IEEE*, 86(11), November 1998.

B. G. Malkiel. *Efficient Market Hypothesis.* Macmillan, London, 1987. (reference found in Lawrence et al. [1996]).

R. D. Martin. Garch modeling of time-varying volatilities and correlations, 1998. URL `http://fenews.com/1998/Issue4/059802.htm`. (URL accessed on March 4, 2002).

T. Masters. *Practical Neural Network Recipes in C++.* Academic Press, New York, 1993. (reference found in Kaastra and Boyd [1996]).

A. McCallum. Rainbow, 1998. URL `http://www-2.cs.cmu.edu/~mccallum/bow/rainbow/`. (URL accessed on April 1, 2002).

J. L. McClelland, D. E. Rumelhart, and the PDP Group. *Parallel Distributed Processing. Explorations in the Microstructure of Cognition*, volume 1: Foundations. MIT Press, Cambridge, MA, 1986. (reference found in Dorffner [1996]).

L. Mendelsohn. Training neural networks. *Technical Analysis of Stocks and Commodities*, pages 40–48, November 1993.

M. Minsky and S. Papert. *Perceptrons*. MIT Press, Cambridge, MA, 1969. (reference found in [Mitchell, 1997, p. 127]).

T. M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

M. Møller. *Efficient Training of Feed-Forward Neural Networks*. PhD thesis, Computer Science Department, Aarhus University, DK-8000 Århus, Denmark, July 1993. URL `http://www.daimi.au.dk/PB/464/PB-464.pdf`.

D. P. Morgan and C. L. Scotfield. *Neural Networks and Speech Processing*. Kluwer Academic Publishers, Boston, 1991. (reference found in Dorffner [1996]).

MSN Money. FTSE 100, 2002a. URL `http://moneycentral.msn.com/investor/glossary/glossary.asp?TermID=1113`. (URL accessed on April 25, 2002).

MSN Money. Leverage, 2002. URL `http://moneycentral.msn.com/investor/glossary/glossary.asp?TermID=267`. (URL accessed on April 25, 2002).

MSN Money. Technical analysis, 2002b. URL `http://moneycentral.msn.com/investor/glossary/glossary.asp?TermID=515`. (URL accessed on April 25, 2002).

P. M. Narendra and K. Fukunaga. A branch and bound algorithm for feature subset selection. *IEEE Transactions on Computers*, 26(9):917–922, 1977. (reference found in [Bishop, 1996, p. 307]).

R. F. Nau. Statistical Forecasting, 2000. URL `http://www.duke.edu/~rnau/411out00.html`. (URL accessed on November 11, 2001).

M. M. Nelson and W. T. Illingworth. *A Practical Guide to Neural Nets*. Addison Wesley, Reading, MA, 1991. (reference found in Kaastra and Boyd [1996]).

M. D. Odom and R. Sharda. A neural network model for bankruptcy prediction. In *Proc. IEEE Int. Conf. on Neural Networks*, pages II163–II168, San Diego, 1992. (reference found in Kaastra and Boyd [1996]).

P.Hürzeler. Konstantin eduardowitsch ziolkowski, 2001. URL `http://mitglied.lycos.de/space_udssr/htm/ziolkowski.htm`. (URL accessed on April 1, 2002).

D. S. G. Pollock. Economic forecasting, 1992. URL `http://www.qmw.ac.uk/~ugte133/courses/tseries/8idntify.pdf`. (URL accessed on October 13, 2001).

R. F. Port, F. Cummins, and J. D. McAuley. Naive time, temporal patterns and human audition. In R. F. Port and T. van Gelder, editors, *Mind as Motion*. MIT Press, Cambridge, MA, 1995. (reference found in Dorffner [1996]).

W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, second edition, 1992. (reference found in [Bishop, 1996, p. 272]).

A. N. Refenes, M. Azema-Barac, L. Chen, and S. A. Karoussos. Currency exchange rate prediction and neural network design strategies. *Neural Computing and Applications*, 1(1), 1991. (reference found in Dorffner [1996]).

S. Rementeria, J. Oyanguren, and G. Marijuan. Electricity Demand Prediction Using Discrete-Time Fully Recurrent Neural Networks. In *Proceedings of WCNN'94*, CA, USA, 1994. (reference found in Dorffner [1996]).

S. Roberts and L. Tarassenko. The Analysis of the Sleep EEG using a Multilayer Neural Network with Spatial Organisation. In *IEE proceedings Part F*, volume 6, pages 420–425, 1992. (reference found in Dorffner [1996]).

A. Robinson. Box Jenkins Methodology, 1999. URL `http://www.bath.ac.uk/~masar/math0118/forecasting/node14.html`. (URL accessed on October 15, 1999).

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning Internal Representations by Error Propagation. In D. E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing, Explorations in the Microstructure of Cognition*, volume 1: Foundations. MIT Press, Cambridge, MA, 1986. (reference found in Dorffner [1996]).

D. Ruppert. GARCH models, 2001a. URL `http://www.orie.cornell.edu/~davidr/or473/LectNotes/notes/node139.html`. (URL accessed on November 11, 2001).

D. Ruppert. Univariate Time Series Models, 2001b. URL `http://www.orie.cornell.edu/~davidr/or473/LectNotes/notes/node31.html#SECTION00500000000000000000`. (URL accessed on October 20, 2001).

N. Schraudolph and F. Cummins. Introduction to neural networks, 2002. URL `http://www.icos.ethz.ch/teaching/NNcourse/backprop.html#top`. (URL accessed on January 23, 2002).

M. Seul, L. O'Gorman, and M. J. Sammon. *Practical Algorithms for Image Analysis.* Cambridge University Press, 2000.

R. Sharda and R. B. Patil. A connectionist approach to time series prediction: An empirical test. In G. J. Deboeck, editor, *Trading on the Edge: Neural, Genetic, and Fuzzy Systems for Chaotic Financial Markets*, pages 451–464. Wiley, New York, 1994. (reference found in Kaastra and Boyd [1996]).

P. S. Simard, B. Victori, Y. LeCun, and J. Denker. Tangent prop – A formalism for specifying selected invariances in an adaptive network. *Advances in Neural Information Processing Systems*, 4, 1992. (reference found in Mitchell [1997]).

C. Siriopoulos, R. N. Markellos, and K. Sirlantzis. Applications of artificial neural networks in emerging financial markets. In A.-P. Refenes, Y. Abu-Mustafa, J. Moody, and A. Weigend, editors, *Neural Networks in Fincancial*

*Engineering, Proc. of the 3rd Int. Conf. on Neural Networks in the Capital Markets*, pages 284–302, Singapore, 1996. World Scientific. (reference found in Hellström and Holmström [1998]).

J. S. Smith. NFL Neural Network Applet, 2000. URL `http://www.softtechdesign.com/software/NFLnet/NFLnet.htm`. (URL accessed on April 25, 2002).

K. A. Smith and J. N. D. Gupta. Neural networks in business: techniques and applications for the operations researcher. *Computers & Operations Research*, 27:1023–1044, 2000.

D. A. Sprecher. On the structure of continous functions of several variables. In *Transactions of the American Mathematical Society*, volume 115, pages 340–355, 1965. (reference found in [Bishop, 1996, p. 137]).

K. Swingler. Financial prediction, some pointers, pitfalls, and common errors, 1994. URL `http://www.geocities.com/francorbusetti/neuralpitfalls.pdf`. (URL accessed on April 26, 2002).

U. F. Systems. Tutorial introduction, 2002. URL `http://www.ultrafs.com/introtut.htm`. (URL accessed on April 25, 2002).

Z. Tang, C. de Almedia, and P. A. Fishwick. Time series forecasting using neural networks vs. Box-Jenkins Methodology. In *International Workshop on Neural Networks*, Auburn, AL, January 1990. (reference found in Kaastra and Boyd [1996]).

P. Terna. Abcde: Agent based chaotic dynamic emergence, 2000. URL `http://www.google.com/url?sa=U&start=1&q=http://web.econ.unito.it/terna/deposito/mabs.pdf&e=921`. (URL accessed on March 31, 2002).

The Numerical Algorithms Group Ltd. Chapter 29: Time series analysis, 2000. URL `http://www.nag.co.uk/numeric/fn/manual/pdf/c09/c09int_fn04.pdf`. (URL accessed on January 25, 2002).

L. Thing and M. Rouse. Expert system, 2002. URL `http://whatis.techtarget.com/definition/0,,sid9_gci212087,00.html`. (URL accessed on April 25, 2002).

J. D. Thomas and K. Sycara. Integrating genetic algorithms and text learning for financial prediction. In A. A. Freitas, W. Hart, N. Krasnogor, and J. Smith, editors, *Data Mining with Evolutionary Algorithms*, pages 72–75, Las Vegas, Nevada, USA, 8 2000. URL `http://citeseer.nj.nec.com/308554.html`.

P. Tino, C. Schittenkopf, and G. Dorffner. Temporal pattern recognition in noisy non-stationary time series based on quantization into symbolic streams: Lessons learned from financial volatility trading, 2000. URL `http://citeseer.nj.nec.com/tino00temporal.html`. (URL accessed on March 30, 2002).

W. G. Tomek and S. F. Querin. Random processes in prices and technical analysis. *J. Futuers Markets*, 4:15–23, 1984. (reference found in Kaastra and Boyd [1996]).

R. Tompkins. *Options Explained*$^2$. Macmillan Press, 1994.

G. Tsibouris and M. Zeidenberg. Testing the efficient market hypothesis with gradient descent algorithms. In A. Refenes, editor, *Neural Networks in the Capital Markets*. John Wiley and Sons, 1995. (reference found in Lawrence et al. [1996]).

C. Ulbricht. Multi-Recurrent Networks for Traffic Forecasting. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pages 883–888, Cambridge, MA, 1994. AAAI Press/MIT Press. (reference found in Dorffner [1996]).

C. Ulbricht. *State Formation in Neural Networks for Handling Temporal Information.* PhD thesis, Institut fuer Med. Kybernetik u. AI, Univ. Vienna, 1995. (reference found in Dorffner [1996]).

C. Ulbricht, G. Dorffner, and A. Lee. Forecasting fetal heartbeats with neural networks. In *Proceedings of the Engineering Applications of Neural Networks (EANN) Conference*, 1996. (reference found in Dorffner [1996]).

University of Derby in Austria. Research Methods in Computing, Student Handout Mastercopies, 2001.

A. Waibel. Consonant Recognition by Modular Construction of Large Phonemic Time-delay Neural Networks. *Advances in Neural Information Processing*, pages 215–223, 1989. (reference found in Dorffner [1996]).

A. S. Weigend, D. E. Rumelhart, and B. A. Huberman. Backpropagation, Weight Elimination and Time Series Prediction. *Connectionist Models*, pages 105–116, 1990. (reference found in Dorffner [1996]).

H. White. Economic prediction using neural networks: The case of ibm daily stock returns. In *IEEE International Conference on Neural Networks, San Diego*, pages 451–459, San Diego, 1988. (reference found in Hellström and Holmström [1998]).

H. White. Economic Prediciton Using Neural Networks. In R. R. Trippi and E. Turban, editors, *Neural Networks in Finance and Investing*, pages 315–328. Probus, Chicago, 1993. (reference found in Dorffner [1996]).

G. Widrow and M. Hoff. Adaptive switching circuits. In *Convention Record*, pages 96–104. Institute of Radio Engineers, Western Electronic Show and Convention, 1960.

P. M. Williams. A marquardt algorithm for choosing the step-size in backpropagation learning with conjugate gradients. Technical Report CSRP-299, Cognitive Science, University of Sussex, Brighton, UK, 1991. (reference found in Møller [1993]).

R. J. Williams and D. Zipser. Experimental Analysis of the Real-time Recurrent Learning Algorithm. *Connection Science*, 1(1):87–111, 1989. (reference found in Dorffner [1996]).

B. Wong, T. Bodnovich, and Y. Selvi. Neural network applications in business: a review and analysis of the literature (1988-1995). *Decision Support Systems*, 19:301–320, 1997b.

B. Wong, T. Bodnovich, and Y. Selvi. Neural network applications in business ; a review and analysis of the literature (1988-95), 1997a. URL `http://www.comp.nus.edu.sg/~cs5264/neural-app.ppt`. (URL accessed on February 28, 2002).

B. Wüthrich, D. Permunetilleke, S. Leung, V. Cho, J. Zhang, and W. Lam. Daily prediction of major stock indices from textual www data, 1998. URL `http://citeseer.nj.nec.com/117658.html`. (URL accessed on April 1, 2002).

M. Zekic. Neural network applications in stock market predictions - a methodology analysis, 1998. URL `http://www.efos.hr/eng/academic_staff/mzekic/mzekic_varazdin98.pdf`. (URL accessed on February 28, 2002).

A. Zell. Answer to inquiry concerning current stage of development of TACOMA. (personal communication), 2002.

# Glossary

## A

**Adaptive Linear Neuron (ADALINE)** Early neural network developed by Widrow and Hoff (Widrow and Hoff [1960]) capable of modelling linearly separable target functions.

**Akaike's Information Criterion (AIC)** Akaike's information criterion is a method for determining the order of an autoregressive model by minimizing an information theoretic function of the model order $p$, AIC($p$) (Castiglioni [2001a]).

**ANN** Artificial Neural Network.

**Autocorrelation** If the disturbance term (unpredictable random component) is not independent of its values in other observations in the time series, it is said to be subject to autocorrelation (serial correlation).

## B

**Backpropagation (Error backpropagation)** Popular training algorithm for artificial neural networks of the multi-layer perceptron type, in which the approximation error of the neural network is propagated backwards from the output, over hidden to input layer.

**Business cycle** The business cycle is the short-term fluctuation of total (economic) output around its trend path. The trend path of output

is the smooth path it follows in the long run once the short-term fluctuations are averaged out (Begg et al. [1997]).

**Buy and hold trading strategy** A long-term investing strategy in which money is invested in buying a portfolio and holding it for a certain period of time, without any reaction to short-term fluctuations of the portfolio's value.

## C

**CG** Conjugate Gradient.

**Computational Neuroscience** Field of computer science concerned with realistic, biologically consistent modelling of the activities in human or animal nervous system by means of computer simulations.

**Confidence interval** The confidence interval is the range, the actual (not the estimated ones) values of a probability distribution are expected to lie.

**Correlation** Measure of association between two variables (comparable to covariance). Correlation, compared to covariance, has the advantage of being independent of units in which the variables are measured (Dougherty [1992]).

**Covariance** Measure of association between two variables (Dougherty [1992]).

**Curse of dimensionality** Phenomenon of decreasing performance of a neural network as reaction to presence of too many inputs. The performance deterioration is contrary to the intuitive assumption that more input data would yield better performance.

## D

**DAX** DAX (Deutscher Aktienindex) is a German benchmark stock index, which is made up of the stock values of 30 leading German companies.

**Delta neutral trading strategy** Option contracts trading strategy, in which the profitability of a trading action depends on the volatility of the underlying commodity, but does not depend on changes in price of the underlying commodity. In such trading strategies (e.g. long straddle, short straddle), the estimation of volatility is crucial for making profitable decisions.

**Dow Jones Industrial Average (DJIA)** The DJIA is computed by adding all the daily stock prices of a group of 30 major US corporations and dividing that total by a number called the divisor. This divisor will change whenever one of the 30 companies declares a stock split. A company will split its stock when the price becomes high, making it more affordable. By changing the divisor, the index value is unaffected by stock splits (Systems [2002]).

## E

**Economics** The study of how human societies decide what, how and for whom to produce (Begg et al. [1997]).

**Eigenvalue** An eigenvalue of an $N \times N$ matrix $A$ is a scalar $c$ such that $Ax = cx$ holds for some nonzero vector $x$ (Herman [1996]).

**Eigenvector** An eigenvector of a $N \times N$ matrix $A$ is a nonzero vector $x$ such that $Ax = cx$ holds for some scalar c (Herman [1996]).

**EMH** Efficient Market Hypothesis.

**Excess profit** Profit of a firm over and above what provides its owners with a normal (market equilibrium) return to capital (Deardorff [2001]).

**Expected value** The expected value of a discrete random variable is the weighted average of all its possible values, taking the probability of each outcome as a weight. In case of a real random variable, the expected value equals to population mean of this variable (Dougherty [1992]).

**Expert system** An expert system is a computer program that simulates the judgement and behavior of a human or an organization that has expert knowledge and experience in a particular field. Typically, such a system contains a knowledge base containing accumulated experience and a set of rules for applying the knowledge base to each particular situation that is described to the program. Sophisticated expert systems can be enhanced with additions to the knowledge base or to the set of rules. Among the best-known expert systems have been those that play chess and that assist in medical diagnosis (Thing and Rouse [2002]).

## F

**Feed-forward neural network** A neural network which in which the nodes (neurons) of the network are organized in groups (layers). All neurons of a layer have input connections only from the previous layer and have output connections only to the next layer, and to no other nodes. The name of this type of neural network stems from the direction of information processing within a trained network - from the input layer forward to hidden layer (or layers) and finally to the output layer.

**FTSE 100 (Financial Times Stock Exchange 100)** The FTSE 100 is a benchmark index tracking the performance of the London Stock Exchange. FTSE 100 comprises the 100 largest companies traded on the exchange (MSN Money [2002a]).

## G

**Gradient** Gradient is the vector of first partial derivatives of a function.

## H

**Heteroskedasticity** Property of a time series, in which the disturbance term (the unpredictable random component) has different probability distribution at different points in time (Dougherty [1992]).

## K

**Kronecker delta symbol** Kronecker delta symbol $\delta_{ij}$ equals to 1, if $i = j$ and equals to 0 otherwise.

## L

**Learning rate** In backpropagation learning algorithm, the learning rate is a measure for the degree to which the weights of a neural network are changed during each training step (Mitchell [1997]).

**Leverage** Leverage is the use of debt to increase the returns of a company (MSN Money).

**Linear separability** Property of a learning problem, in which a line or plane can be drawn that separates all training instances into the appropriate classes.

## M

**Macroeconomics** Macroeconomics is a discipline of economics concerned with investigation of the economy as a whole (as opposed to microeconomics which offers a detailed treatment of individual decisions about particular commodities). It deliberately simplifies the individual building blocks of the analysis in order to retain a manageable analysis of the complete interaction of the economy (Begg et al. [1997]).

**MLP** Multilayer Perceptron.

**Momentum term** In variations of the original backpropagation algorithm, the momentum term establishes a partial dependence of the weight change at a certain learning step upon the weight change at the previous learning step. This is done for the purpose of overcoming the capture (oscillation) of the learning algorithm in local minima of the error function or in flat regions of the error function. It also has the effect of gradually increasing the step size of the search in regions where the gradient is unchanging, thereby speeding convergence (Mitchell [1997]).

**Multi-layer perceptron (MLP)** An artificial neural network possessing one or more hidden layers.

## N

**Neural network** A neural network is an interconnected assembly of simple processing elements whose functionality is modelled after the neuron in the brain. The processing capability of the network is determined by the relative strengths (weights) of these connections. These weights are calculated by the process of adaption to (and learning from) a set of training patterns (Smith [2000]) .

## O

**Option contract** An option contract gives the holder the right, but not the obligation to buy or sell a previously specified commodity or stock (underlying commodity) at a previously specified price at a previously specified point in time in the future.

## R

**Random walk** Time series of observations, where each observation is a sum of the previous observation plus a random number.

**Residual** The difference between actual and estimated (by means of a prediction technique) value of a time series (Dougherty [1992]).

## S

**SCG** Scaled Conjugate Gradient.

**Serial correlation** See *Autocorrelation*.

**Sliding window technique** In order to account for the time-varying nature of many time series, prediction algorithms are trained not on the whole data set, but instead on a small subset (sliding window size) of the data which correspond to a certain period of time prior to the point in time to be predicted. The in this way trained algorithm is applied for making predictions for a few days ahead. Then, the sliding window is shifted by a small number, the algorithm is retrained and again applied to make predictions a few days ahead.

**Standard and Poor's index (S&P 500 Index)** S&P 500 Index is made up of 500 stocks. The total market value of each company is computed and summed. A company's market value is equal to the share price times the total shares outstanding. The sum of all 500 companies' market values is then divided by a divisor. The result is that each company influences the index based on its total market value rather than its share price. This type of index is a capitalization-based index (Systems [2002]).

**Stationarity** Property of a time series in which probability distributions involving values of the time series are independent of time translations (Castiglioni [2001b]).

**Stock Market Index** A stock market index is a number computed from the prices of a group of stocks. It is computed daily to gauge the movement in the market for that day (Systems [2002]).

## T

**TACOMA (TAsk decomposition by COrrelation Measures)** Neural network system which automatically determines the optimal network topology for the task to be learned (automatic topology determination), thus relieving the user of the need to determine the optimal topology empirically.

**Technical analysis** The practice of trying to divine stock prices by examining trading patterns and comparing the shape of current charts to those from the past (MSN Money [2002b]).

**Test set** Part of the available data set which is used to test the performance of an adaptive prediction algorithm. Usually train and test sets are chosen not to overlap in order to get a measure of generalization ability of the algorithm.

**Training set** Part of the available data set which is used to train an adaptive prediction algorithm.

## V

**Volatility** Volatility is the synonym for standard deviation of some quantity (e.g. stock price) within a defined period (e.g. during a day). As is the case with delta neutral trading strategies, sometimes correct estimation of volatility is crucial for making profitable trading decisions.

## W

**Weights** Strengths of the connections between individual nodes in an artificial neural network. By changing of these strengths (weights) the network can be "trained" to produce arbitrary output pattern as reaction to a certain input pattern.

**White noise** Time series of randomly distributed real numbers with zero mean and no association between observations drawn at different points of time.

# Appendix A

# AR example

The example of fitting an AR(1) process to a log-return time series of daily General Electric average stock prices presented here is developed by Ruppert in Ruppert [2001b].

## A.1 Notation

We follow Ruppert's notation in this example, thus defining the AR(1) process as

$$
\begin{aligned}
y_t - \mu &= \phi(y_{t-1} - \mu) + \varepsilon_t \\
y_t &= (1 - \phi)\mu + \phi y_{t-1} + \varepsilon_t
\end{aligned}
$$

## A.2 Example

Ruppert fits the process to a time series proceeding in following steps:

1. Transformation of prices to log prices
2. Transformation of log prices to log returns
3. Estimation of parameters of the AR process
4. Calculation of the SACF of the residuals

Log returns are defined as

$$\Delta p_t$$

where $p$ is the time series of log prices. Using the multiple regression analysis (see appendix B), the parameters of the AR(1) process are estimated:

$$\begin{aligned}
&y_t = (1-\phi)\mu + \phi y_{t-1} + \varepsilon_t \\
&\hat{y} = a + bx \\
&y = y_t \\
&x = y_{t-1} \\
&a = (1-\phi)\mu \\
&b = \phi \\
&b = \frac{\mathrm{Cov}(y_{t-1}, y_t)}{\mathrm{Var}(y_{t-1})}
\end{aligned}$$

The regression operation is supported by popular statistics software programs like SPSS or SAS. The regression yields following values

$$\begin{aligned}
&a = -0.0000907 \\
&b = 0.22946
\end{aligned}$$

Now the basic statistical test has to be performed:

$$H_0 : \phi = 0 \Rightarrow b = 0 \quad H_1 : \phi \neq 0 \Rightarrow b \neq 0$$

The null hypothesis states that the log returns of the stock prices are just white noise, i.e. there is no correlation with past values. The regression results are tested with the t test:

$$\begin{aligned}
&s_u^2 = \frac{n}{n-2}\,\mathrm{Var}(e) \\
&\mathrm{s.\,e.}(b) = \sqrt{\frac{s_u^2}{n\,\mathrm{Var}(x)}} \\
&\mathrm{s.\,e.}(b) = 0.062 \\
&t = \frac{b-\beta_0}{\mathrm{s.\,e.}(b)} \\
&t = \frac{0.22946-0}{0.062} \\
&t = 3.7010
\end{aligned} \tag{A.1}$$

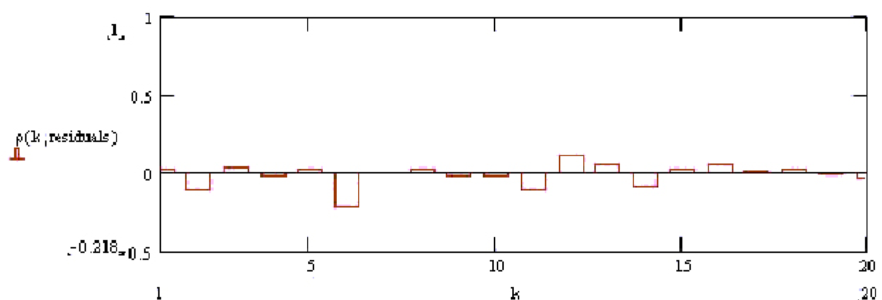



Figure A.1: SACF of the residuals

We check the null hypothesis at 5 % significance level with a two-tailed t test with 249 degrees of freedom (number of log returns minus number of parameters estimated, in our case 2, a and b). The critical value of t is in this case equal to 1.97. We reject our null hypothesis ($b = \phi = 0$) since the t statistic does not lie within the interval $[-1.97, 1.97]$. Hence the time series being analysed is not a white noise (at least an AR(1) model fits it better than white noise).
But does an AR(1) model actually fit the time series well? If the model predicts the time series well, then the residuals should actually be unpredictable, i.e. random, not autocorrelated values.

We analyse that using the SACF function of the residuals (see figure A.1).

Note the huge outlier at the point $k = 6$. This is a hint that in fact, there is an autocorrelation in the residual values. Hence, the AR(1) is not a proper means for modelling this time series.

# Appendix B

# Regression analysis

Multiple regression analysis is a technique for finding a function of type

$$y = \alpha + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_k x_k + u \tag{B.1}$$

which represents a model of relationship between two or more variables ($x_1$ to $x_k$). $u$ is the so-called *disturbance term*, i.e. the non-predictable, random component.

Consider the observations of the variables x and y shown in figure B.1:

The task of the regression analysis is to fit a line such that the sum of squared residuals is minimized (see figure B.2). A residual is the difference between the actual and estimated value. In figure B.2 the residuals are represented by broken lines. Given $k = 1$ (simple regression analysis, value of $y$ depends on only 1 variable),

$$\begin{aligned} y &= \alpha + \beta_1 x_1 + u \\ \hat{y} &= a + b x_1 \\ a &= \bar{y} - b\bar{x} \\ b &= \frac{\operatorname{Cov}(x, y)}{\operatorname{Var}(x)} \end{aligned} \tag{B.2}$$

The derivation of these expressions is omitted here and can be found in other sources (e.g. Dougherty [1992]).

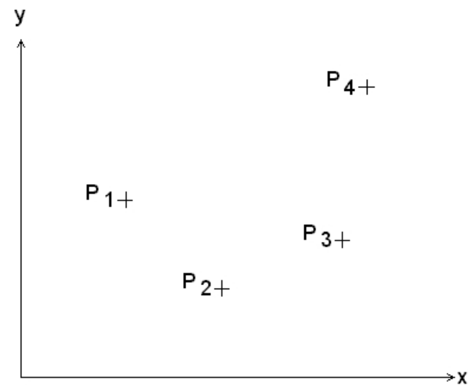



Figure B.1: Observations of variables $x$ and $y$

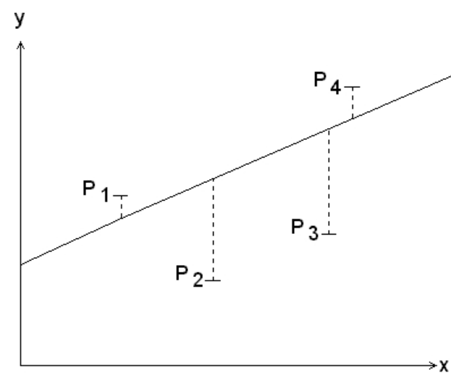



Figure B.2: Fitted line so that the sum of squared residuals is minimized

# Appendix C

# ARIMA example

For the demonstration of the ARIMA model the author uses the Ox software package and the "Cree Incorporated" stock price data.

First, we look at the SACF[1] of this data (the diagram is created using an Ox function, see figure C.1).

The original time series is not stationary, the differences are. Therefore we choose the ARIMA$(p, 1, q)$ model.

Using an Ox script, we estimate the parameters of the ARIMA$(1, 1, 1)$ model and get the following results (output of the Ox script):

```
Coefficient     Std.Error   t-value     t-prob
d parameter           1.00000    (fixed)
AR-1                 0.343038     0.3539    0.969   0.333
MA-1                -0.161787     0.6089   -0.266   0.791
Cree_1             -0.0626317     0.3364   -0.186   0.852

log-likelihood    -625.663391
no. of observations       226  no. of parameters          4
AIC.T              1259.32678  AIC                5.5722424
mean(Cree)            63.0963  var(Cree)            215.906
```

[1]In Ox the function is said to create an ACF diagram, in this context it is assumed that in Ox ACF means SACF.

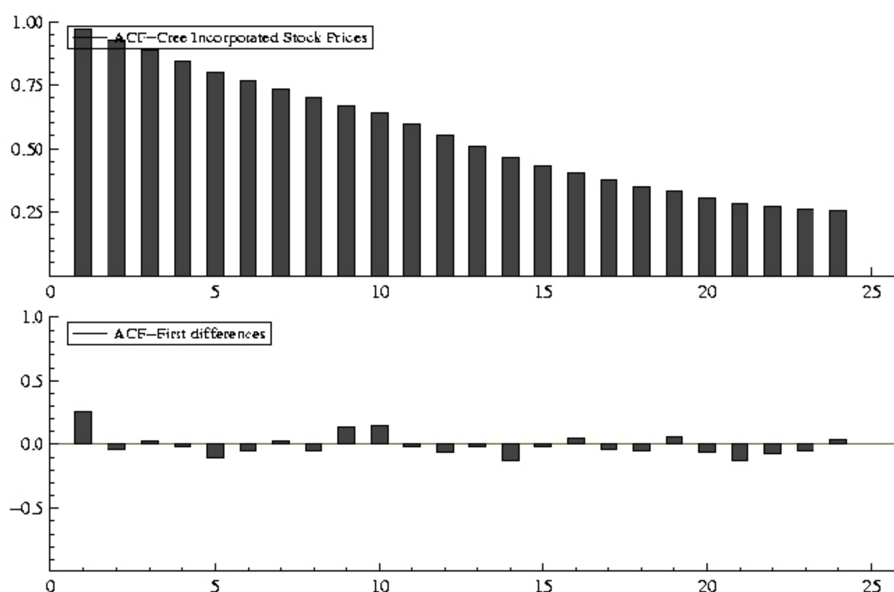



Figure C.1: SACFs of the original time series and the time series after the taking the differences

```
sigma                    3.86399  sigma^2                14.9304
```

The parameters of the ARIMA process are denoted by `AR-1` and `MA-1`. The value `AIC` means *Akaike's information criterion*. This is a measure of how good a model fits the actual data. When comparing multiple models, the one with the least value of AIC is considered best (Ruppert [2001b]).

We try out other models and get the AIC values shown in table C.1.

| Model | AIC |
| --- | --- |
| ARIMA(1,1,1) | 5.5722424 |
| ARIMA(1,1,2) | 5.58066465 |
| ARIMA(0,1,1) = MA(1) | 5.56335283 |
| ARIMA(1,1,0) = AR(1) | 5.56369347 |
| ARIMA(0,1,2) = MA(2) | 5.56002487 |
| ARIMA(2,1,0) = AR(2) | 5.57226212 |

Table C.1: AIC values for different ARIMA models

# Appendix D

# Forecasting with ARIMA

In this example we follow Borchers (Borchers [2001]) and use the ARIMA$(1,1,1)$ model for the Cree Incorporated stock prices from appendix C.

Thus we have following setting:

$$\text{ARIMA}(1,1,1) \quad \gamma_1 = 0.343038 \quad \theta_1 = -0.161787 \tag{D.1}$$

Suppose we want predict the level of stock price 4 elements ahead (i.e. we want to know the values $y_{n+1}$ till $y_{n+4}$ where $n$ is the number of the last element of the "training" set, i.e. the last element of the part of the time series which was used for calculating the coefficients of the ARIMA model).

Now we write out the model in the mixed form:

$$\begin{aligned}
&(1-\gamma_1\,\mathrm{B})(1-\mathrm{B})y_t = (1-\theta_1\,\mathrm{B})\varepsilon_t \\
&(1-0.343038\,\mathrm{B})(1-\mathrm{B})y_t = (1-0.161787\,\mathrm{B})\varepsilon_t \\
&(1-\mathrm{B}-0.343038\,\mathrm{B}+0.343038\,\mathrm{B}^2)y_t = \varepsilon_t - 0.161787\,\mathrm{B}\,\varepsilon_t \\
&(1-1.343038\,\mathrm{B}+0.343038\,\mathrm{B}^2)y_t = \varepsilon_t - 0.161787\,\mathrm{B}\,\varepsilon_t \\
&y_t - 1.343038\,\mathrm{B}\,y_t + 0.343038\,\mathrm{B}^2\,y_t = \varepsilon_t - 0.161787\,\mathrm{B}\,\varepsilon_t
\end{aligned} \tag{D.2}$$

Now we substitute the backwards operator by his meaning (see section 1.4.1, p. 25). As we know $\mathrm{B}\,x$ means the value of element preceding $x$.

$$y_t - 1.343038y_{t-1} + 0.343038y_{t-2} = \varepsilon_t - 0.161787\varepsilon_{t-1} \tag{D.3}$$

Hence our actual coefficients are:

$$\gamma_1 = -1.343038 \quad \gamma_2 = 0.343038 \quad \theta_1 = -0.161787$$

Now it is time to calculate the $\psi$ weights. Their number depends on the number of forecasts we desire to make (in our case 10). The transfer function is equal to

$$\begin{aligned} \psi(\mathrm{B}) &= \frac{\theta(\mathrm{B})}{\phi(\mathrm{B})} \\ \psi(\mathrm{B}) &= \frac{(1 - 0.161787\,\mathrm{B})}{(1 - 1.343038\,\mathrm{B} + 0.343038\,\mathrm{B}^2)} \end{aligned} \tag{D.4}$$

The author uses a computer program to perform the expansion of the Taylor series and obtains the results shown in figure D.1. Therefore we have following

$$\psi(B) := \frac{1 + 0.161787 \cdot B}{1 - 1.343038 \cdot B + 0.343038 \cdot B^2} = \begin{aligned} &1 + 1.504825 \cdot B + 1.677999158350 \cdot B^2 + 1.737404475282067300 \cdot B^3 \ldots \\ &+ \blacksquare + 1.7577827563918098025 \cdot B^4 + 1.7647732811871336510 \cdot B^5 + \blacksquare \ldots \\ &+ 1.7671712968318719534 \cdot B^6 + 1.7679939073226116912 \cdot B^7 + \blacksquare \ldots \\ &+ 1.7682760939801340694 \cdot B^8 + 1.7683728947267572310 \cdot B^9 + \blacksquare \ldots \\ &+ 1.7684061010612773471 \cdot B^{10} + O\left(B^{11}\right) \end{aligned}$$

Figure D.1: Taylor series expansion

$\psi$ weights:

$$\psi_1 = 1 \quad \psi_2 = 1.504825 \psi_3 = 1.677999158350 \quad \psi_4 = 1.737404475282067300$$

Now it is time to calculate the values $\varepsilon_t$ for the elements of the "training" set. This is done using the simple script shown in figure D.2. Having the values $\varepsilon_t$ at his disposal, the author proceeds to the calculation of the forecasts and corresponding confidence intervals (figure D.3).

The results are presented in table D.1.

From these figures one notes that the "uncertainty" of the prediction increases as we move further, i.e. the more elements ahead we predict, the more uncertain are these predictions. As Niels Bohr said, *Prediction is very difficult, especially if it's about the future.*

$\gamma 1 := -1.343038 \qquad \gamma 2 := 0.343038 \qquad \theta 1 := -0.161787$

getNull(row, column) := 0

t := 2 .. trainingSetSize

```
calculateErrors := | ε ← matrix(trainingSetSize, 1, getNull)
                   | ε_{0,0} ← y_{0,0}
                   | ε_{1,0} ← y_{1,0} + γ1·y_{0,0} + θ1·ε_{0,0}
                   | for  t ∈ 2 .. trainingSetSize
                   |   ε_{t,0} ← y_{t,0} + γ1·y_{t-1,0} + γ2·y_{t-2,0} + θ1·ε_{t-1,0}
                   | ε
```

Figure D.2: Script for the calculation of the $\varepsilon_t$ time series

```
| for  i ∈ trainingSetSize + 1 .. trainingSetSize + 1 + forecastsNumber
|   y_{i,0} ← γ1·y_{i-1} + γ2·y_{i-2} + ε_i + θ1·ε_{i-1}
| y
```

Figure D.3: Script for the calculation of the confidence intervals

| **Prediction** | **Actual value** | **CI (Upper)** | **CI (Lower)** | $\sigma_t^2$ |
|---|---|---|---|---|
| -42.675 | 37.188 | -31.019 | -54.331 | 35.364 |
| 67.874 | 34.281 | 177.573 | -41.825 | 55.97 |
| -105.796 | 32.500 | 35.963 | -247.555 | 72.327 |
| 165.372 | 37.500 | 334.021 | -3.277 | 86.047 |

CI (Upper) = Upper confidence interval

CI (Lower) = Lower confidence interval

Both confidence intervals at 95 %

Table D.1: Confidence intervals

# Appendix E

# Taylor series expansion

As explained in Gregg [2001], the Taylor series are a tool for calculating the value of a function to an arbitrary level of precision (in a certain region). Briefly said, with a Taylor series one can approximate every function $f(x)$ by using the following technique:

$$\begin{aligned} f[x] &= \sum_{n=0}^{\infty} a_n (x-a)^n \\ a_n &= \frac{f^{(n)}[a]}{n!} \end{aligned} \tag{E.1}$$

where $f^{(n)}$ means the n-th derivative of the function $f(x)$. Such a series is called Taylor series centered at $x = a$. The accuracy of this approximation decreases as $x$ moves away from the initial $a$ value. Therefore the optimal $a$ value depends on the region where the function is to be estimated.

# Appendix F

# Confidence intervals

The confidence interval is the range, we expect the actual values (not the estimated ones) to lie. The 95 % confidence interval, for example, is the range the actual values will fall in with a probability of 95 %. The confidence intervals are given by

$$\mu + z\sigma \quad \text{and} \quad \mu - z\sigma \tag{F.1}$$

where $z$ is the value of the inverse cumulative probability density function. The inverse cumulative density function tells us, how many standard deviations a value can maximally be away from the estimated value to within a certain confidence interval.

Look at the figure F.1.

The green lines mark the area (this is the area between the green lines) within which observations will lie with a probability of 95 %. The area outside the green lines correspond to values, whose probability of occurrences is less than 5 % (the so-called *tails*). The inverse cumulative probability density function (*qnorm*) gives us the $Z$ number, i.e. the number of standard deviations a value may lie away from the mean value to be within the 95 % confidence interval. Since the probability density curve is symmetric and we are not interested in deviations in a certain direction (i.e. the probability that a value is higher or smaller), we have to take the two-tailed version and have to determine the inverse cumulative probability for the 97,5 % value (that is because we have 5

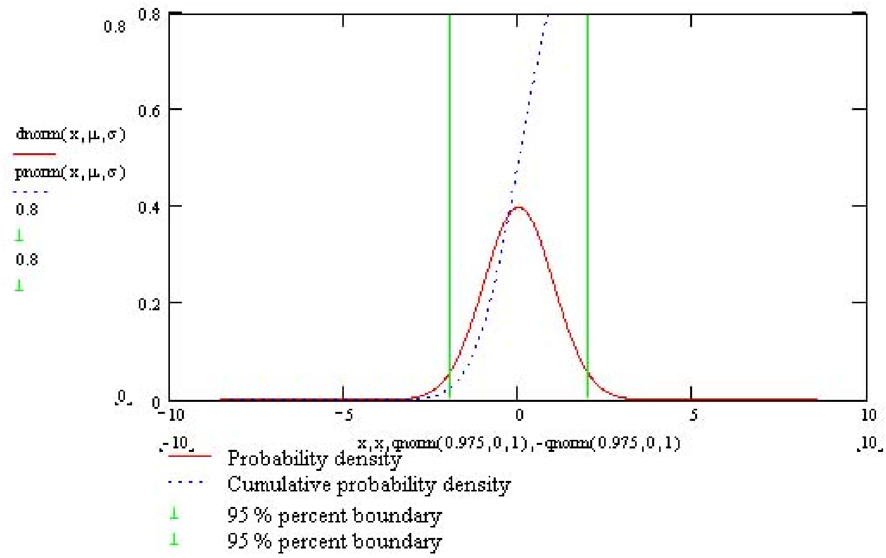



Figure F.1: Probability density and the confidence intervals

% tails on both sides, and only 2,5 % on each particular side).

In the case of the 95 % confidence interval we have a $Z$ value of 1.96, ie the boundaries of values lying within the 95 % probability region are

$$\mu + 1.96\sigma \quad \text{and} \quad \mu - 1.96\sigma \tag{F.2}$$

# Appendix G

# Kolmogorov's theorem

The information found in this section was taken from [Bishop, 1996, pp. 137–141].

There is a theorem due to Kolmogorov (Kolmogorov [1957]) which, although of no direct practical significance, does have an interesting relation to neural networks. The theorem has its origins at the end of the nineteenth century when the mathematician Hilbert compiled a list of 23 unsolved problems as a challenge for twentieth century mathematicians (Hilbert [1900]). Hilbert's thirteenth problem concerns the issue of whether functions of several variables can be represented in terms of superpositions of functions of fewer variables. He conjectured that there exist continuous functions of three variables which cannot be represented as superpositions of functions of two variables. The conjecture was disproved by Arnold (Arnold [1957]). However, a much more general result was obtained by Kolmogorov (Kolmogorov [1957]) who showed that every continuous function of several variables (for a closed and bounded input domain) can be represented as the superposition of a small number of functions of one variable. Improved versions of Kolmogorov's theorem have been given by Sprecher (Sprecher [1965]), Kahane (Kahane [1975]) and Lorentz (Lorentz [1976]). In neural network terms this theorem says that any continuous mapping $y(x)$ from $d$ input variables $x_i$ to an output variable $y$ can be represented exactly by a three-layer neural network having $d(2d + 1)$ units in the first hidden layer and

$(2d + 1)$ units in the second hidden layer.

Without saying, how to obtain the appropriate weights (thus of little practical significance), the theorem states that *every* function can theoretically be approximated by a MLP with two second layers. So it is a well-founded counter-argument to the work of Minsky and Paper (Minsky and Papert [1969]) since it shows that multi-layer networks (contrary to single-layer networks addressed by Minsky and Papert) *are* able to represent all functions very well. So Kolmogorov's theorem (although developed in times where ANNs were not used widely) contributed to promoting ANN related research to a high degree.

# Appendix H

# The $\mathcal{O}$ notation

The $\mathcal{O}$-notation is a theoretical measure of the execution of an algorithm, usually the time or memory needed, given the problem size $n$, which is usually the number of items. Informally, saying some equation

$$f(n) = \mathcal{O}(g(n)) \tag{H.1}$$

means $f(n)$ is less than some constant multiple of $g(n)$(Black [2002]).
Formally, $f(n) = \mathcal{O}(g(n))$ means there are positive constants $c$ and $k$, such that

$$0 \leq f(n) \leq c \times g(n) \text{ for all } n \geq k.$$

The values of $c$ and $k$ must be fixed for the function $f$ and must not depend on $n$ (Black [2002]).
The importance of this measure can be seen in trying to decide whether an algorithm is adequate, but may just need a better implementation, or the algorithm will always be too slow on a big enough input. For instance, quicksort, which is $\mathcal{O}(n \log n)$ on average, running on a small desktop computer can beat bubble sort, which is $\mathcal{O}(n^2)$, running on a supercomputer if there are a lot of numbers to sort. To sort 1,000,000 numbers, the quicksort takes 6,000,000 steps on average, while the bubble sort takes 1,000,000,000,000 steps (Black [2002]).
Any measure of execution must implicitly or explicitly refer to some computation model. Usually this is some notion of the limiting factor. For one problem

or machine, the number of floating point multiplications may be the limiting factor, while for another, it may be the number of messages passed across a network. Other measures which may be important are compares, item moves, disk accesses, memory used, or elapsed (”wall clock”) time (Black [2002]).

# Appendix I

# Fisher's discriminant

## I.1 Derivation for 2-classes case

Fisher's linear discriminant[1] is a tool for reduction of dimensionality of the input data while preserving as much of the class discriminatory information as possible. This happens by projecting $N$ $d$-dimensional samples $x_1, \ldots, x_N$, which belong to classes $\mathcal{C}_1, \mathcal{C}_2$ onto a line

$$y = w^T x \tag{I.1}$$

such that the class separation is maximized. Consider figure I.1. In this simple case there are two features ($x_1$ and $x_2$, $d = 2$), hence the projection results in the reduction of dimensionality of the data. Using a good projection it is possible to discriminate both classes with only one ($y$) instead of two ($x_1, x_2$) features. In order to achieve a maximal class separation, a measure thereof must be introduced. Theoretically, the difference between the mean values of classes in the projection could be used for this purpose. Such a measure would be equal to

$$\mu_i = \frac{1}{N_i} \sum_{x \in \mathcal{C}_i} x \quad \tilde{\mu}_i = \frac{1}{N_i} \sum_{y \in \mathcal{C}_i} y = \frac{1}{N_i} \sum_{x \in \mathcal{C}_i} w^T x = w^T \mu_i \tag{I.2}$$

$$J(w) = |\tilde{\mu}_1 - \tilde{\mu}_2| = |w^T(\mu_1 - \mu_2)| \tag{I.3}$$

[1] The derivation of Fisher's discriminant follows Gutierrez-Osuna [2001].

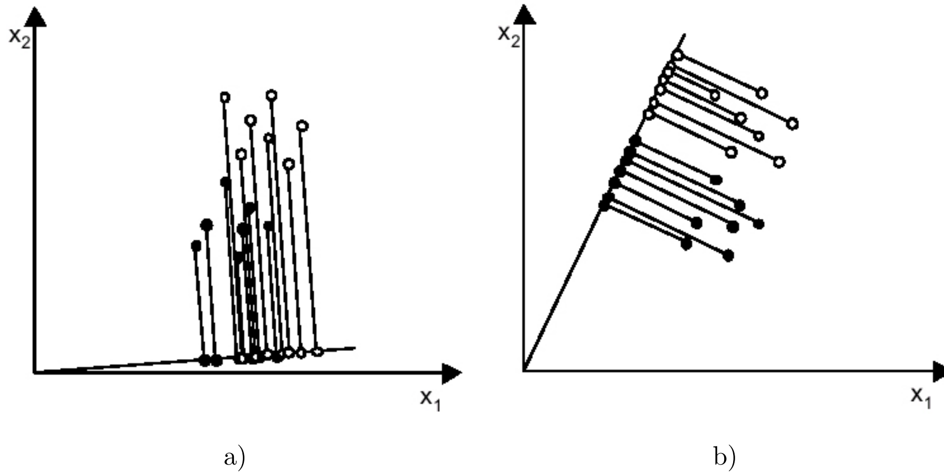



Figure I.1: Two projections of training samples onto a line. The samples indicated by empty and filled circles belong to two different classes. The projection b) yields a better class separation (discrimination) than a) (Gutierrez-Osuna [2001]).

But the distance between projected means is not a good measure for class separation since it does not take into account the standard deviation within the classes (figure I.2). The solution proposed by Fisher is to maximize a function that represents the difference between the means, normalized by a measure of the within-class scatter. For each class, a *scatter* $\tilde{s}_i^2$ (equivalent of the variance) is defined:

$$\tilde{s}_i^2 = \sum_{y \in \mathcal{C}_i} (y - \tilde{\mu}_i)^2 \tag{I.4}$$

The quantity $\tilde{s}_1^2 + \tilde{s}_2^2$ is called the *within-class-scatter* of the projected examples. The Fisher linear discriminant is defined as the linear function $w^T x$ that maximizes the criterion function

$$J(w) = \frac{|\tilde{\mu_1} - \tilde{\mu_2}|^2}{\tilde{s}_1 - \tilde{s}_2} \tag{I.5}$$

In other words, the desired projection is the one, where examples from the same class are projected very close to each other and, at the same time, the projected means are as farther apart as possible. In order to find $w$ that maximizes $J(w)$, $J(w)$ must be expressed as an explicit function of $w$. For this purpose, several

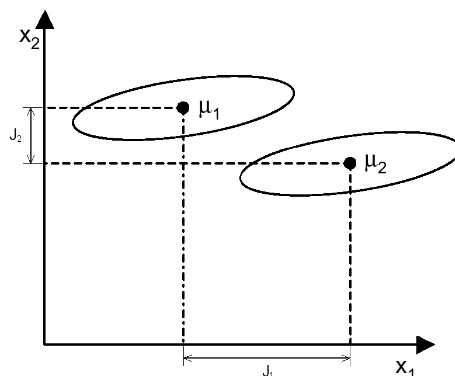



Figure I.2: Projection on $x_2$ axis yields a better class separation in spite of the fact that separation on $x_1$ axis would result in a larger distance between means, since there is a bigger overlap between classes on the $x_1$ axis ($\mu_1$ and $\mu_2$ are means of classes $\mathcal{C}_1$ and $\mathcal{C}_2$) than on the $x_2$ axis. The standard deviation (ellipses) should be taken into account when deriving a measure for class separability (Illustration from [Bishop, 1996, p. 107]).

new definitions are necessary:

$$S_i = \sum_{x \in \mathcal{C}_i} (x - \mu_i)(x - \mu_i)^T \tag{I.6}$$

$$S_1 + S_2 = S_W \tag{I.7}$$

The *within-class scatter matrix* $S_W$ is proportional to the sample covariance matrix. The scatter of the projection can be expressed as a function of the scatter matrix:

$$\tilde{s}_i^2 = \sum_{y \in \mathcal{C}_i} (y - \tilde{\mu}_i)^2 = \sum_{x \in \mathcal{C}_i} (w^T x - w^T \mu_i)^2 \tag{I.8}$$

$$\tilde{s}_i^2 = \sum_{x \in \mathcal{C}_i} w^T (x - \mu_i)(x - \mu_i)^T w = w^T S_i w \tag{I.9}$$

$$\tilde{s}_1^2 + \tilde{s}_2^2 = w^T S_w w \tag{I.10}$$

In the same way, the numerator in equation I.5 can be redefined as

$$(\tilde{\mu_1} - \tilde{\mu_2})^2 = (w^T\mu_1 - w^T\mu_2)^2 = w^T(\mu_1 - \mu_2)(\mu_1 - \mu_2)^T w = w^T S_B w \quad \text{(I.11)}$$

$$S_B = (\mu_1 - \mu_2)(\mu_1 - \mu_2)^T \quad \text{(I.12)}$$

$$\Rightarrow (\tilde{\mu_1} - \tilde{\mu_2})^2 = w^T S_B w \quad \text{(I.13)}$$

$S_B$ is called the *between-class scatter*. Finally, Fisher criterion is defined in terms of $S_W$ and $S_B$ as

$$J(w) = \frac{w^T S_B w}{w^T S_w w} \quad \text{(I.14)}$$

In order to find the $w^*$ such that $J(w)$ is maximized, the first derivation of $J(w)$ must be equated to zero:

$$\frac{d}{dw}(J(w)) = 0 \quad \text{(I.15)}$$

$$\frac{d}{dw}\left(\frac{w^T S_B w}{w^T S_W w}\right) = 0 \quad \text{(I.16)}$$

$$\frac{\frac{(w^T S_B w)}{dw}(w^T S_W w) - \frac{(w^T S_W w)}{dw}(w^T S_B w)}{(w^T S_W w)^2} = 0 \quad \text{(I.17)}$$

$$\frac{(w^T S_B w)}{dw}(w^T S_W w) - \frac{(w^T S_W w)}{dw}(w^T S_B w) = 0 \quad \text{(I.18)}$$

$$2S_B w(w^T S_W w) - 2S_W w(w^T S_B w) = 0 \quad \text{(I.19)}$$

$$S_B w(w^T S_W w) - S_W w(w^T S_B w) = 0 \quad \text{(I.20)}$$

Dividing by $(w^T S_W w)$ yields

$$S_B w - \frac{S_W w(w^T S_B w)}{(w^T S_W w)} = 0 \quad \text{(I.21)}$$

$$J(w) = \frac{w^T S_B w}{w^T S_w w} \quad \text{(I.22)}$$

$$\Rightarrow S_B w - J(w) S_W w = 0 \quad \text{(I.23)}$$

$$S_B w = J(w) S_W w \quad \text{(I.24)}$$

$$\frac{S_B}{S_W} = J(w) \quad \text{(I.25)}$$

The solution of the last expression yields

$$w^* = \arg_w \max\left(\frac{w^T S_B w}{w^T S_W w}\right) = S_W^{-1} S_B w(\mu_1 - \mu_2) \quad \text{(I.26)}$$

This is known as Fisher's Linear Discriminant (Fisher [1936]). Strictly speaking, it is not a discriminant but rather a specific choice of direction for the projection of the data down to one dimension.

## I.2 Generalized version of Fisher's discriminant

The equation I.26 applies to problems with 2 classes. The general version, applicable to arbitrary numbers of classes will be given on following lines.
In this case, there is not 1, but $(C-1)$ discriminant functions, $C$ being the number of classes. The generalization of the within-class scatter matrix is

$$S_W = \sum_{i=1}^{C} S_i \tag{I.27}$$

$$S_i = \sum_{x \in \mathcal{C}_i} (x-\mu_i)(x-\mu_i)^T \quad \mu_i = \frac{1}{N_i} \sum_{x \in \mathcal{C}_i} x \tag{I.28}$$

The generalization for the between-class scatter matrix is

$$S_B = \sum_{i=1}^{C} N_i(\mu_i - \mu)(\mu_i - \mu)^T \tag{I.29}$$

$$\mu = \frac{1}{N} \sum_{n=1}^{N} x_n = \frac{1}{N} \sum_{k=1}^{C} N_k \mu_k \tag{I.30}$$

The matrix

$$S_T = S_B + S_W \tag{I.31}$$

is called the *total scatter matrix*.
The task of the "generalized" Fisher's discriminant is to find $(C-1)$ projection vectors $w_i$, which can be arranged by columns into a projection matrix $W = [w_1|w_2|\dots|w_{C-1}]$ so that

$$y_i = w_i^T x \Rightarrow y = W^T x \tag{I.32}$$

The mean vector and scatter matrices for the projected samples are defined as

$$\tilde{\mu}_i = \frac{1}{N_i} \sum_{y \in \mathcal{C}_i} y \quad \tilde{\mu} = \frac{1}{N} \sum_{n=1}^{N} y_n \tag{I.33}$$

$$\tilde{S}_W = \sum_{i=1}^{C} \sum_{y \in \mathcal{C}_i} (y-\tilde{\mu}_i)(y-\tilde{\mu}_i)^T \quad \tilde{S}_B = \sum_{i=1}^{C} N_i(\tilde{\mu}_i - \tilde{\mu})(\tilde{\mu}_i - \tilde{\mu})^T \tag{I.34}$$

From the derivation for the two-class problem, it can be shown that

$$\tilde{S}_W = W^T S_W W \quad \tilde{S}_B = W^T S_B W \tag{I.35}$$

"Generalized" Fisher's criterion is defined similarly to the two-classes case:

$$J(W) = \frac{\tilde{S}_B}{\tilde{S}_W} = \frac{|W^T S_B W|}{|W^T S_W W|} \tag{I.36}$$

The projection matrix, that maximizes $J(W)$ is called $W^*$. It can be shown that the optimal projection matrix $W^*$ is the one whose columns are the eigenvectors[2] corresponding to the largest eigenvalues[3] of the following generalized eigenvalue problem:

$$W^* = \arg\max \left( \frac{W^T S_B W}{W^T S_W W} \right) \Rightarrow (S_B - \lambda_i S_W) w_i^* = 0 \tag{I.37}$$

[2]An eigenvector of a $N \times N$ matrix $A$ is a nonzero vector $x$ such that $Ax = cx$ holds for some scalar c (Herman [1996]).

[3]An eigenvalue of an $N \times N$ matrix $A$ is a scalar $c$ such that $Ax = cx$ holds for some nonzero vector $x$ (Herman [1996]).