FORECASTING STOCK PRICE MOVEMENTS USING NEURAL NETWORKS

Christian Rank

MSc Thesis Submitted on **14 August 2006**

Word Count: **24 112**

MSc Thesis Presented for the Degree of **Master of Science**, in the Department of Statistical Sciences, University of Cape Town, South Africa.

Supervisor: Professor Renkuan Guo

Dr. Hans-Georg Zimmermann

DECLARATION

Copyright:

I hereby grant the University of Cape Town permission to copy and disseminate this work, or any part thereof, for the purposes of study and research.

Plagiarism:

1. This dissertation is my own work.
2. I have not allowed, and will not allow, anyone to copy my work with the intention of passing it off as his or her own work
3. I have used the American Psychological Association convention for citation and referencing. Each significant contribution to, and quotation in, this dissertation from the work of other people has been cited and referenced.

I am now presenting the thesis for examination for the degree of **Master of Science.**

Christian Rank

Abstract

The prediction of security prices has shown to be one of the most important but most difficult tasks in financial operations. Linear approaches failed to model the non-linear behaviour of markets and non-linear approaches turned out to posses too many constraints. Neural networks seem to be a suitable method to overcome these problems since they provide algorithms which process large sets of data from a non-linear context and yield thorough results.

The first problem addressed by this research paper is the applicability of neural networks with respect to markets as a tool for pattern recognition. It will be shown that markets posses the necessary requirements for the use of neural networks, i.e. markets show patterns which are exploitable. In a second step the reasons for the existence of these patterns is explained in a behavioural context. It can be summarized that there is a large set of social elements which influence the decision making process of a market participant and make him act rather irrationally in contrary to what is postulated by the hypothesis of efficient markets.

Subsequently, the basic theoretical concepts of neural networks are explained. Two specific types of networks, feedforeward and recurrent Networks are defined along with respective sub-types such as Monte Carlo Networks and Error Correction Networks. The introduced types of networks will then be used to simulate market movements and the outcomes are compared to a simple buy-and-hold approach. The application will be based on daily data of ABSA a stock from the Johannesburg Stock Exchange. Different kinds of networks yield different results. A comparative discussion of the diverse outcomes is the recessional part of this paper. The expected result, i.e. recurrent networks yield better results than feed forward networks and Monte Carlo networks only show trends while they are disadvantageous for day trading decisions, can be confirmed. However, all types of networks but Monte Carlo networks outperform a buy-and-hold approach transaction costs respectively.

As a conclusion it can be claimed that neural networks are indeed useful devices for making financial forecasts.

Christian Rank **08-11-2006**

Abstrakt

Die Prognose von Wertpapierpreisen ist eine der essentiellsten jedoch diffizilsten Aufgaben in der Finanzindustrie. Lineare Modelle versagen zu oft das nicht lineare Verhalten von Märkten zu simulieren. Nicht lineare Modelle andererseits besitzen zu viele Restriktionen. Neuronale Netze scheinen adäquate Modelle zu sein, diese Probleme zu umgehen. Sie stellen Algorithmen zur Disposition, die große Mengen von Daten aus nicht linearem Kontext verarbeiten und akkurate Resultate liefern.

Die erste Frage, die diese Arbeit adressiert, ist die Applikabilität neuronaler Netze in Bezug auf Märkte als ein Mittel zur Mustererkennung. Es wird gezeigt, dass der Charakter von Märkten den Einsatz neuronaler Netze erlaubt, d.h. Märkte zeigen in der Tat exploitable Zyklen. Die Existenz solcher Zyklen wird mit Hilfe der „Behavioural Finance Theorie" beleuchtet und es werden eine Reihe sozialer Einflussfaktoren genannt, die irrationales Verhalten von Seiten des Marktteilnehmers erklären, was im Gegensatz zur Theorie effizienter Märkte steht.

Folgend werden theoretische Konzepte neuronaler Netze erklärt. Zwei spezifische Typen von Netzwerken, feedforward und rekurrente Netze werden definiert. Fokus wird dabei auch auf Subtypen wie Monte Carlo Netze und Error Correction Netze gerichtet.

Abschließend soll ein Titel des südafrikanische Finanzmarkts simuliert werden wobei tägliche Daten des Johannesburg Stock Exchange als Basis dienen. Die Aktionsvorschläge der Simulationen sollen mit einem simplen „buy-and-hold" Ansatz verglichen werden. Da diverse Netze differierende Resultate liefern, endet diese Arbeit mit einer komparativen Diskussion der Ergebnisse. Hier bestätigt sich die Annahme, dass rekurrente Netze bessere Resultate liefern als feedforward Netze und dass Monte Carlo Netze zwar gute Trends prognostizieren, aber keine solide Basis für day-trading Entscheidungen darstellen. Alle Netze mit Ausnahme der Monte Carlo Netze, erzielten jedoch bessere Resultate als der simple „buy-and-hold" Ansatz. Daher kann mit Recht behauptet werden, dass neuronale Netze ein wertvolles Mittel zur Entscheidungsfindung im Finanzbereich darstellen.

Christian Rank **08-11-2006**

Acknowledgements

A very interesting lecture on neural networks and their applicability in finance held by Dr. Hans-Georg Zimmermann was the reason for my choosing this field as the topic of my dissertation. At this place I want to thank Dr. Zimmermann for his patience and enthusiasm in bringing this somewhat difficult concept closer to me. Furthermore I want to give my thanks to Professor Renkuan Guo who functioned as my supervisor here in Cape Town. He was always available for me and supported me with ideas, advises and supplied me with data. I also want to thank Professor Cas Troskie who did not shy any expenses in order to provide the necessary software and Mr. Li and Mr. Cui for installing SENN. Finally I am very grateful to Miss Yujia Jiang who always gave me a hand whenever my schedule was tight.

Contents

Figures

Tables

Graphics

1. Introduction

In order to make profits a financial analyst must find the right signals for buying and selling. To do so, he basically uses two different approaches which differ fundamentally, although their final goal – to give a reasonable forecast – is equal. These two concepts are referred to as fundamental and technical analysis.

Fundamental analysts use information about a company to forecast security movements. This information is retrieved by looking at balance sheets, cash flows or income statements. An important role plays the concept of the intrinsic or fair value of a security in order to predict its future behaviour. The fair value of a security is the price, which is justified and free of any bias [Liv99, pp.23]. The task of the fundamental analyst is it to find exactly this price. In comparing the fair value of a company with its market value, i.e. the price which is requested by the market - driven by demand and supply - the fundamental analyst decides whether a security is undervalued or overvalued. If e.g. the market value of a security is less than its fair value, a buy signal is given. The underlying economic assumption is that market prices gradually converge to fair prices over time. Knowing both values, a fundamental analyst is able to predict the direction of a security's future evolving.

Technical analysis on the other hand solely refers to market factors when trying to predict future developments. The underlying assumption here is that price movements follow certain patterns which can be refined in any level of sub-patterns. The technical analyst assumes that patterns occur in defined orders. Subsequently, having identified a certain pattern he is able to forecast a security's future development. Thus he is able to give the right buy and sell signals.

Having analysed the two ways of predicting future prices, we instantly notice their major shortfalls.

Fundamental analysis is a time and resource consuming process relying on the correctness of data and assumptions made. If certain information is not given or falsely stated fundamental analysis leads to spurious conclusions.

The problem of the technical analysis on the other hand is that patterns might not be recognised since they are too subtle or disregarded since the search space is too vast.

In recent years the concept of neural networks as a device of forecasting security prices has appreciated more and more attention in the financial industry. A neural network is an information processing paradigm which is inspired by the way biological nervous systems, like the brain, process information. It is composed of a large number of highly interconnected processing elements (neurons) working unison to solve specific problems. Neural networks, like people, learn by example and are configured for applications such as pattern recognition or data classification [SS96, p.2].

Its property of pattern recognition could be the solution to the problems of the technical analysis. Historical data for numerous securities can be fed into a neural network, along with known results, and the net can learn which parameters indicate that a stock will rise or fall [Zir97, p.18].

The aim of this research paper is the verification of the last statement. The scientific approach of this paper is two-fold. A theory part will introduce the reader to underlying concepts of the topic and give him the necessary knowledge to both understand the software and interpret the results of the practical part. Historical data from the Johannesburg Stock Exchange will be used as inputs to a decision making system called SENN which was developed by SIEMENS AG. The research will eventually verify the further up given hypothesis that stock markets are indeed predictable applying neural networks.

In order to get a profound knowledge of neural networks and its algorithms chapter 3 will give an explanation of how the mathematical concept of a neural network is derived from the biological concept. Besides it will be illustrated how neural networks apply to markets.

As mentioned, neural networks learn by example, i.e. they have to be trained. Chapter 4 introduces three kinds of training. Among these, supervised training is claimed to be the most important and will be the basis on which the most common learning rules are explained. Additional concepts like the error function as subject of the optimization problem and the backpropagation algorithm will be thoroughly described.

Chapter 5 explains the special case of a feed-forward neural network (FNN). After having introduced the basic concept of a FNN, two special subtypes are presented along with a way of how to decrease model uncertainty.
Chapter 6 treats a more advanced type of neural networks which is referred to as a recurrent neural network (RNN). A comparison to FNN is given as well as an explanation of the concept of "time". This is an important feature which contributes to the superiority of RNN in comparison to FNN. Finally the concept of the error correction neural network (ECNN) is discussed.
In chapter 7 neural networks will be applied on the South African financial market and outcomes discussed.

However, before explaining neural networks in detail, evidence shall be given why neural networks can be applied to financial markets. In particular, the question shall be answered why pattern recognition works in line with the widely accepted concept of efficient markets. To do so chapter 2 takes a look at the theory of efficient markets and tries to reveal whether it applies in reality.

2. Applicability of neural network modelling

Once again the strength of neural networks is the recognition of patterns. Price patterns in particular, are defined as graphical figures which seem to appear with certain regularity. This means, once the neural network has identified a certain pattern from past experience it is able to infer the further evolving of a variable and thus give useful support to the analyst in predicting the future.
However, a market must have a certain constitution in order to allow the application of neural networks as a device for pattern recognition. Markets must actually display regular patterns – how subtle ever – in order to apply an appropriate trading strategy once the pattern is identified. The question whether or not markets show regular patterns and whether these patterns can be exploited is subject of this chapter. First we will explain the well accepted theory of efficient markets which rejects the possibility of regular and predictable patterns. We will explain the importance of information and three different kinds of efficiency. In the following we will find evidence that

markets are not as efficient as the theory suggests. The evidence will be based on academic studies which found market anomalies and certain trading strategies outperforming a buy-and-hold approach. We will try to explain these findings with a discipline of science which is called "behavioural finance" and was first introduced in the mid 1980s. Figure 2.1 shows some patterns that frequently occur in the market.

Graphic 2.1 Frequent market patterns [Sc06]

2.1 Efficient markets in theory

The foundation of the efficient market hypothesis was made by Eugene Fama in his famous paper in which he derives the concept and gives a definition of efficient markets [Fam70]. His paper is based on an earlier work in which Fama concluded that stock prices follow a random walk [Fam65, pp.35]. He refers to Harry Roberts who demonstrated that a random walk looks like actual stock series [Rob59, pp.3]. Roberts was also the first to make distinctness between different kinds of efficiencies.

Many different definitions of efficient markets are known among academics. Timmermann uses the definitions of Jensen and Malkiel in his paper on forecasting and efficient markets [TG04, pp.16]. According to these, a market is said to be efficient if it fully and correctly reflects all relevant information in determining security prices. More formally, a market is efficient with respect to information set Ω_t if it is impossible to make economic profits by trading on basis of information set Ω_t.

As mentioned above the concept of efficient markets is based on the random walk theory. The random walk theory states that stock prices only change due to new information. Since the information producing world is very complex and seen as a black box, the appearance of new information is perceived to be random, i.e. unpredictable. Since information is instantaneously incorporated in the security price it is not possible to beat the market using the offered information.

But what is the reason why information is instantaneously incorporated in the price and why is the so generated value of a security supposed to be the right value? The argument is that a large amount of well skilled analysts examine the market using all kinds of information given to them in order to find security prices. This conglomerate of competence is said to be the reason for the exact assessment of a security's value.

According to Roberts, Timmermann distinguishes three levels of market efficiency [TG04, p.17],

- *Weak Efficiency*: It is not possible to outperform a weak efficient market if the information set Ω_t consists on past and current prices. I.e. by using charts and interpreting patterns it is not possible to

consistently beat the market. Yet it is possible to outperform the market using published and insider information,

- *Semi-strong Efficiency*: This form is somewhat more rigorous. Not only the use of past and current prices fails to outperform the market but neither published information can be helpful to succeed. Only insider information is a source of useful input. However, insider trading is illegal and so the last form of efficient markets is rather theoretical but shall yet be mentioned,
- *Strong Efficiency*: In strong efficient markets not even insider information helps to consistently outperform the market. Prices adapt so quickly that investors are reduced to price takers.

To summarise the introduced theory we find three major characteristics. Firstly, all relevant information is instantaneously incorporated in the security's price. Only new information causes price changes. Since news can hardly be predicted, price steps cannot be predicted either. That is the reason why continuously abnormal returns are not possible.
This school of thinking had, and still has an enormous impact on investment decisions of both institutional and private investors. As Malkiel writes in his classic "A Random Walk Down Wall Street" the selection of stocks can be randomly processed. There is no point in committing in research since all information is already priced. He draws the picture of picking by darting on a stock record [Mal73]. Even more recent studies reveal that professionally managed portfolios rarely outperform the market.
It becomes clear that in efficient markets, irrespective of the level, price patterns do not exist and thus the use of neural networks for pattern recognition would be nothing but squandered resources.

2.2 Efficient markets in practice

In order to find evidence that markets are indeed exploitable by using neural networks as a means, the following two points must be verified.
Inefficient markets result from market anomalies. Firstly, if anomalies and their reasons can be found regarding real data and secondly, if evidence about successful trading strategies arising from market inefficiency exist (i.e.

strategies that exploit these patterns), we can claim that the preconditions for the use of neural networks are given. Figure 2.2 illustrates the thought process.

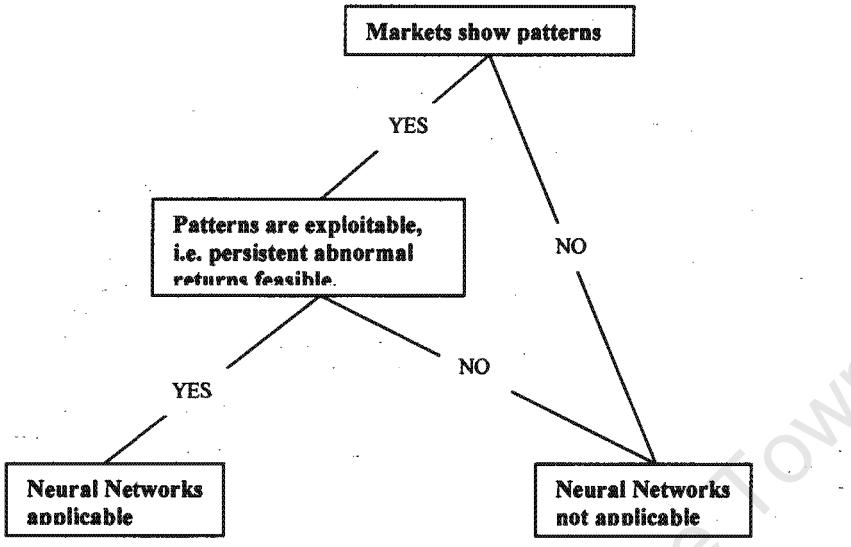


Figure 2.1 Preconditions for the sensible use of neural networks

2.2.1 Anomalies and market patterns

To fully understand the issue, it is important to know how inefficient markets look like.

Inefficient markets are constructs where prices adjust slowly so that profit opportunities through arbitrage exist. Such markets are characterised by anomalies which are defined as price distortions due to structural factors like unfair competition and lack of transparency or behavioural biases by economic agents [Idc06]. Literature distinguishes many kinds of anomalies [Dim88] whereas the most important ones are technical anomalies and fundamental anomalies.

Fundamental anomalies can be referred to information concerning a company's constitution which arise from balance sheets, income or cash flow statements. Technical anomalies address the time series of financial data.

Inefficient markets in a fundamental sense are characterised by the slow dispersion of information among investors. If information spreads slowly, the major postulation of the efficient market hypothesis - the instantaneous adjustment of prices incorporating all information - is violated.

Inefficient markets in a technical sense are characterised by autocorrelations in the time series of security prices. Under the term autocorrelation we understand the correlation of a variable with itself over a substantial period. This means that the security price at time t influences the security price at time t+1. We distinguish two kinds of autocorrelation. The first one is called trending or momentum. Its characteristic is it that the direction of the second variable is more often than not like the direction of the first variable. Trending can be positive and negative. The second kind of autocorrelation is the mean reversion, where the direction of the second variable is more often different than alike the trend of the first variable.

Autocorrelation is the extreme opposite of the random walk theory which states that price steps – or variables of a time series – are independent of one another.

As already mentioned, the theory of efficient markets is based on the random walk theory. If it is now possible to collect enough evidence that security prices show autocorrelations, i.e. they depict patterns over time, we can reject the theory that markets are efficient and so use neural networks as an advanced tool for pattern recognition.

Until the late 1970s the bulk of the investment world has completely absorbed and fully accepted the idea of efficient markets and acted accordingly when constructing portfolios. In 1978 however, Michael Jensen published a paper in which he summarised the more important studies concerning efficient markets and anomalies [Jen78]. His paper contains studies of stock and option markets for periods from the late 1940s till the mid 1970s. Many researchers mentioned in Jensen's paper actually found substantial non-zero abnormal returns but rather contributed their findings to the inadequacy of the underlying asset pricing model than to market inefficiency [Jen78, p.98]. Jensen speaks of a general reluctance to reject the notion of market efficiency but also highlights the fact that anomalies exist and that further studies shall reveal new knowledge [Jen78, p.100].

In 1981 LeRoy and Porter showed that excess volatilities found in the market reject efficiency [LP81, p.558]. Excess volatility is a term for price fluctuations of securities which cannot be explained by fundamentals, i.e. information about dividends or earning announcements. LeRoy and Porter concluded

that there must be another factor apart from fundamental information that drives security prices. Shiller comes to the same result [Shi81, p.421]. He notes that security prices move too much to be justified by subsequent changes in dividends. As an example he shows that during the Great Depression in the late 1920s and early 1930s dividends and earnings where not substantially low although volatility was enormous [Shi81, p. 422].

The year 1985 marks an important milestone in the history of efficient market theory. Like their colleagues four years earlier, DeBondt and Thaler also discovered excess volatilities in security prices but contributed them to overreaction by market agents [DT85]. Their paper marked the beginning of a completely new scientific approach to the problem of efficient markets which was later called "behavioural finance". Yet another important result of their findings was that markets seem to show substantial evidence for weak form inefficiency. Since weak form inefficiency implies patterns in time series, DeBondt's and Thaler's research was crucial for the application of neural networks in forecasting. One year later in 1986 Fischer Black introduced the concept of noise traders and manifests the still young field of behavioural finance. Noise traders usually do not trade on basis of information (rational trading) but simply follow market trends [Bla86, pp.1]. In 1988 Lo and MacKinlay published a paper in which they examined the random walk theory applied to financial markets. Using simple specification tests based on variance estimators they found significant positive autocorrelation for weekly and monthly returns [LM88, p.42]. The examined time period reached from 1962 to 1985. The authors justified their choice of weekly data with the sufficient amount of information to get significant explanation power while minimising biases inherent in daily samples [LM88, p.50]. Lo and MacKinlay also examined the influence of infrequent trading as a source of autocorrelation. Incorporating a Bernoulli trial for the probabilities of trading and non-trading they came to the result that non-trading is not the main reason for autocorrelation [LM88, p.59].

Yet this strong evidence against efficient markets was not enough to completely shift paradigm. In 1991 Fama wrote an extensive paper in which he defends the theory of efficient markets against the challenges which have emerged throughout the last decade. One major difficulty of rejecting the

theory was named to be the problem of the joint-hypothesis. According to Fama it is not evident whether anomalies indicate market inefficiency or whether they are rather due to shortfalls in the underlying asset pricing model [Fam91, p.1576]. Fama also addresses Lo and MacKinlay's paper from 1988. He claims that correlations are rather due to their reducing of variance by diversification than to inefficiencies. He also challenges the Bernoulli approach to incorporate non-trading [Fam91, p.1579]. Concerning the recent appearance of behaviouralists he admits that the theory is intriguing but doubts any proof power [Fam91, p.1583]. As a last point Fama attacks the volatility tests conducted by LeRoy ten years earlier. In his opinion these tests were not informative since they were only based on returns and did not include other important parameters [Fam91, p.1586]. Similar arguments are named in a paper by Malkiel from 2003 in which he defends efficient markets [Mal03].

Nevertheless a very recent paper by Anderson, Eom, Hahn and Park confirms price adjustments over time and so again challenges the theory of efficient markets. The authors used direct tests rather than indirect ones - as applied before – in order to eliminate non-synchronous trading and the bid-ask bounce [AEHP05, p.1]. As a second innovation they tested autocorrelation averaged on single stocks and stocks grouped by firm size [AEHP05, p.2]. The result yielded positive weekly and intraday correlations for small and medium sized companies over a period of ten years. For large companies they found partly positive, partly negative correlations [AEHP05, p.5]. Also stock portfolios were found to show positive correlations [AEHP05, p.6].

The last three decades of research on efficient markets reveal that it is difficult both to absolutely support and to reject the theory of efficient markets. Too many valid points of discussion were introduced by both camps. For Timmermann markets are temporarily inefficient. Only within this period arbitrageurs might gain considerable profits [TG04, p.21]. Keane introduces another interesting aspect. He argues that for some agents markets are more efficient than for others. He refers this fact to their proper investment skills [Kea83, pp.25]. This gives the proof that markets indeed have periods of inefficiency and that market agents with certain skills – e.g.

knowledge of the use of neural networks – have realistic prospects in predicting the market. The question however, remains whether or not these anomalies are indeed exploitable as the theory denies.

2.2.2 Exploitation of patterns

The hypothesis of efficient markets implies that there is no trading strategy that generates abnormal returns over a rather long period. A trading strategy is an exactly defined way in which securities are bought and sold. Trading strategies are long-term oriented. Wärneryd depicts six different kinds of strategies [Wär02, pp.55],

- *Indexing*: the choice of investment is taken in the way that the average returns follow a predefined benchmark,
- *Diversification*: this strategy implies that securities of one category or different categories are mixed in a portfolio in order to reduce systematic risk,
- *Stop and Loss*: the investor defines limits which inevitably lead to buys or sells,
- *Relative Strength*: stocks that have previously outperformed are taken into the portfolio and stock that have underperformed are excluded,
- *Contrarian*: this strategy is opposite of the relative strength. Stocks are bought that have previously underperformed and those are sold which outperformed the market,
- *Technical Analysis:* Trends and their reversals are detected and investments placed accordingly.

Investors use trading strategies to exploit market anomalies. In efficient markets however, none of these strategies should yield abnormal results. Now if there is evidence to be found that all or at least some of these strategies provide satisfying outcomes over a long period we have found the final proof that markets show inefficient behaviour and that they can be beaten. In the following some evidence for exploitation using the last three strategies will be presented.

theory was named to be the problem of the joint-hypothesis. According to Fama it is not evident whether anomalies indicate market inefficiency or whether they are rather due to shortfalls in the underlying asset pricing model [Fam91, p.1576]. Fama also addresses Lo and MacKinlay's paper from 1988. He claims that correlations are rather due to their reducing of variance by diversification than to inefficiencies. He also challenges the Bernoulli approach to incorporate non-trading [Fam91, p.1579]. Concerning the recent appearance of behaviouralists he admits that the theory is intriguing but doubts any proof power [Fam91, p.1583]. As a last point Fama attacks the volatility tests conducted by LeRoy ten years earlier. In his opinion these tests were not informative since they were only based on returns and did not include other important parameters [Fam91, p.1586]. Similar arguments are named in a paper by Malkiel from 2003 in which he defends efficient markets [Mal03].

Nevertheless a very recent paper by Anderson, Eom, Hahn and Park confirms price adjustments over time and so again challenges the theory of efficient markets. The authors used direct tests rather than indirect ones - as applied before – in order to eliminate non-synchronous trading and the bid-ask bounce [AEHP05, p.1]. As a second innovation they tested autocorrelation averaged on single stocks and stocks grouped by firm size [AEHP05, p.2]. The result yielded positive weekly and intraday correlations for small and medium sized companies over a period of ten years. For large companies they found partly positive, partly negative correlations [AEHP05, p.5]. Also stock portfolios were found to show positive correlations [AEHP05, p.6].

The last three decades of research on efficient markets reveal that it is difficult both to absolutely support and to reject the theory of efficient markets. Too many valid points of discussion were introduced by both camps. For Timmermann markets are temporarily inefficient. Only within this period arbitrageurs might gain considerable profits [TG04, p.21]. Keane introduces another interesting aspect. He argues that for some agents markets are more efficient than for others. He refers this fact to their proper investment skills [Kea83, pp.25]. This gives the proof that markets indeed have periods of inefficiency and that market agents with certain skills – e.g.

knowledge of the use of neural networks – have realistic prospects in predicting the market. The question however, remains whether or not these anomalies are indeed exploitable as the theory denies.

2.2.2 Exploitation of patterns

The hypothesis of efficient markets implies that there is no trading strategy that generates abnormal returns over a rather long period. A trading strategy is an exactly defined way in which securities are bought and sold. Trading strategies are long-term oriented. Wärneryd depicts six different kinds of strategies [Wär02, pp.55],

- *Indexing*: the choice of investment is taken in the way that the average returns follow a predefined benchmark,
- *Diversification*: this strategy implies that securities of one category or different categories are mixed in a portfolio in order to reduce systematic risk,
- *Stop and Loss*: the investor defines limits which inevitably lead to buys or sells,
- *Relative Strength*: stocks that have previously outperformed are taken into the portfolio and stock that have underperformed are excluded,
- *Contrarian*: this strategy is opposite of the relative strength. Stocks are bought that have previously underperformed and those are sold which outperformed the market,
- *Technical Analysis:* Trends and their reversals are detected and investments placed accordingly.

Investors use trading strategies to exploit market anomalies. In efficient markets however, none of these strategies should yield abnormal results. Now if there is evidence to be found that all or at least some of these strategies provide satisfying outcomes over a long period we have found the final proof that markets show inefficient behaviour and that they can be beaten. In the following some evidence for exploitation using the last three strategies will be presented.

Jegadeesh and Titman examined the exploitation of relative strength and contrarian strategies. They used a data series ranging from 1965 to 1989 and actually found abnormal returns over a relatively large period of time. According to their paper from 1993 applying the strategy of relative strength - i.e. buying past winners – works favourably up to the first seven months. Later on a contrarian strategy which is buying past losers yields abnormal returns [JT93, p.67]. In order to give maximal strength to the test, it was designed to avoid any bias from bid-ask spreads, price pressure or lagged reaction effects [JT93, p.68]. In addition Jegadeesh and Titman referred these abnormal return generating strategies to patterns in the markets which are created by under- and overreacting market agents [JT93, p.90]. One of the major critiques of the paper was that the results could have been forced by data snooping since the subject of examination was only the US market. To rule out this possibility Rouwenhorst conducted the same test in an international context. Rouwenhorst engineered country-neutral portfolios and found that controlling for country composition average excess returns were only slightly diminished [Rou98, p. 273]. He concluded that country momentum was relatively unimportant in explaining the success of these strategies [Rou98, p.275]. This gives evidence that at least two of the three strategies under examination do indeed yield abnormal returns.

In 1997 Bessembinder and Chan tested 26 technical trading rules as strategies to obtain abnormal returns. They discovered that there was forecast power using technical analysis and that it was not attributable to measurement errors [BC97, p.3]. Even including trading costs did not substantially diminish profits from these strategies [BC97, p.14]. Although Bessembinder and Chan did not conclude that markets are inefficient [BC97, p.18] it is yet interesting that technical rules can lead to abnormal returns.

In conclusion it can be said that markets show all preconditions for the use of neural networks. Evidence was given that markets show anomalies, i.e. patterns. Secondly, evidence was given that there are indeed strategies to continuously exploit patterns. The question however, how these patterns come to existence and why markets are temporarily inefficient has not yet been discussed. The next section tries to provide some answers and is crucial for understanding the formation of market prices. The field of

behavioural finance can help to give a decent explanation of what causes temporary deviations from rationality.

2.3 Behavioural finance

To understand why markets often behave contrarian to the efficient market hypothesis we have to take a closer look at market agents. Markets do not have a proper behaviour. They just reflect the behaviour of their participants. This seemingly slight difference is crucial for the on going examination. In order to comprehend how markets react we have to understand how investors and other market participants derive their decisions. Since all market participants are human beings we have to take a look at the psychological side.

A central determinant of how investors behave depends on their expectations of the future development of a market. There are only guesses about future events. These guesses are partly based on experience and market agents believe that they can use their experience in order to assess future events irrespective of their uncertainty. On the other hand guesses also depend on new information which reaches the market agent at any stage. These more or less founded guesses are called expectations and may change as new information arrives.

Wärneryd introduces two kinds of expectations [Wär02, pp.73]. *Mathematical expectation* can be described as a distribution with mean and variance. A probability is attached to every possible outcome so that the market value – the true value of a security - represents a weighted average over all potential results.

Psychological expectation on the other hand comprises possible actions and outcomes which may differ from mathematical expectation. This expectation however, is the base on which individual market participants take decisions.

In the following we describe how psychological expectation is formed and why this leads to markets which are not in accordance with efficient market postulations.

Wärneryd suggests that expectation is a weighted sum of believes based on the past (extrapolation) P_t, believes based on the discrepancy between

extrapolation and real outcomes (learning) L_t and new information I_t [Wär02, pp.81]. The formation of psychological expectations can thus be written as

$$EXP_{t+1} = w_1 P_t + w_2 L_t + w_3 I_t , \qquad (2.1)$$

where w_i represent specific weights with $\Sigma_i w_i = 1$.

Depending on how much trust a decision taker has in new information and how he is influenced by past experience the weights will be set appropriately. Rationality means, as the efficient market hypothesis claims that all information is rationally used. In reality however, this is not the case. The formation of expectations is heavily influenced by cognitive processes and emotions [Wär02, pp.82]. In the following we will show how the processing of new information in particular is biased. In order to process information we need a certain input. This input can be biased since human beings tend to leave out relevant information. This is referred to as selective attention. Mass media and communication devices produce so many data that only little of it can be handled at the same time. This phenomenon is called the cognitive bias. DeBondt showed that cognitive bias is responsible for investors to detect patterns in random series or to identify wrong patterns [DT98, p.833]. Social influence like herd behaviour, the attention to market rumours and the necessity of out guessing other traders also lead to biased input [Wär02, pp.216]. In addition unstructured information and time pressure do also contribute to the bias.

Not only biased information processing but also emotions like mood or confidence play an important role in expectation formation. Every human being is exposed to swaying emotions and it should be of no question to acknowledge that emotions strongly influence our behaviour. Mood affects the degree of which we optimistically interpret information [Wär02, pp.161]. Market participants of good mood rather tend to be more optimistic about the future and interpret data more positive than they actually are. The opposite can be observed if an investor's mood is rather bad. Confidence on the other hand is an important factor of expectation formation, too [Wär02, pp.167]. Confidence affects the weights (w_i) we assign to our expectation function. An investor who has good experience with past assessments and strategies will

rather put more weight on extrapolation than on new information. But there are also investors who rather integrate news in the decision process. It should be mentioned here that it is rather typical that human beings easily forget about the past and mostly work with new information. In this case, the weights of the expectation function would be set rather differently.

Another important factor which influences investors' behaviour is social imitation. In his paper on imitation in social psychology, Ellenwood declares that all social activities be it in daily life or science are somehow outcomes of imitation [Ell01, p.722]. Social imitation is a learning process. Miller and Dollard distinguish three kinds of imitation [MD41, pp.76].

- *Same behaviour*: persons perform the same act in response to independent stimulation by the same cue, each having learned by himself to make the response. An example would be the taking of the same bus.
- *Copying*: a person learns to model its behaviour on that of another one. The copier knows when their behaviour is the same.
- *Matched-dependent behaviour*: occurs when one person is older, shrewder or more skilled and others match their behaviour dependent on his.

The first two are not applicable to investment decisions since markets are too complex for "same behaviour" and "copying" is difficult since investment details are treated confidentially. The third point however, can be observed when analysing investors' actions and so it is subject of the proceeding explanation.

In certain situations human beings are dependent on the guidance of a stronger part [Mos01, p.259]. Experiments with children, looking for hidden candies survey that the younger child orients its search on the places the older one has chosen [MD41, p.78]. This observation is also called a suggestion-imitation process [Ell01, p.722]. Imitative behaviour however, is not only restricted to childhood. In social life such as financial markets are, individuals are being placed in analogous situations as described [MD41, p.81]. Imitative behaviour can be observed in each social field. Subsequently,

imitation is due to the perception of uncertainty and the desire of safety and orientation in a complex environment [MD41, pp.183] such as financial markets. According to Peter Lynch, one of the most successful portfolio managers professional investors are pardoned on loosing money as long as they loos with securities popular and accepted in the market [Arn02, p.9]. Loosing with IBM shares is regarded as less dramatic than loosing with stocks of hardly known companies. If a substantially large group of investors has the same expectation about the development of a security, it acts as the strong part described in the experiment. Individual investors as the weaker part have learnt that going with, i.e. imitating the market's opinion means being on the safe side in cases of loss. Thus it is proven that imitating is a common phenomenon in investments and that it results from a learning process as stated above. Wärneryd subsumes imitation under the term herd behaviour [Wär02].

Self control as a last point of influence shall be mentioned. Self control is referred to as a protection from ill-considered actions. Its dimensions can be summarised as foresight, i.e. adequate paying attention to the future and willpower, i.e. control of present enjoyment [Wär02, pp.190]. These two dimensions can be subsumed under rational behaviour. Both having an investment plan which defines future milestones and the necessary care not to loose everything due to greed are essential to market participants. Unfortunately human beings tend to loose focus on their goals. Sometimes stock rallies are too tempting and investors join the bandwaggon without considering the consequence of potential losses. Especially investment firms and funds which have to submit their results in very short intervals in order to inform their clients, often have to digress from long run strategies to achieve immediate positive results.

The following graphic summarises what influences psychological expectation in real life and what leads to the deviation from rationality.

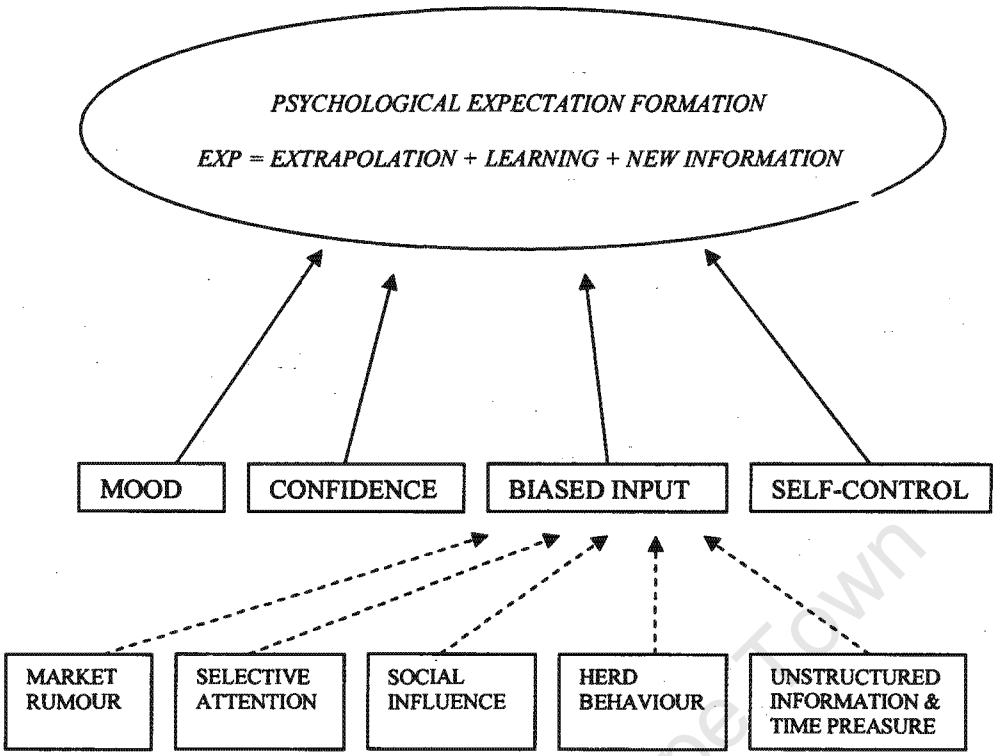


Figure 2.2 Factors that influence psychological expectations

To summarise we note that markets do not have any behaviour but they are rather indicators of the behaviour of their participants. Market agents base any action on their proper expectation of the future. Rather than applying the mathematical form of expectation formation – which the theory of efficient markets postulates - individuals naturally process psychological expectation building. This kind of expectation is a weighted sum of experience, learning and integration of new information. Human beings are subject to moods, properties of character, social influence or accessibility of information. This and many more factors influence our expectation formation and lead to a often considerable deviation between guesses and actual outcomes. Since this is proven fact, it can be claimed that market participants act far from rational and thus markets do partly not yield prices which conform to the efficient market theory. The two forms of deviation are overreaction and underreaction [Wär02, pp.155]. The general concept is illustrated in the following graphic where p is the market price of a security, p^* denotes the fair price according to an event and t is the time with t_0 denoting the moment in which an event occurs, i.e. the information about the event starts to spread.

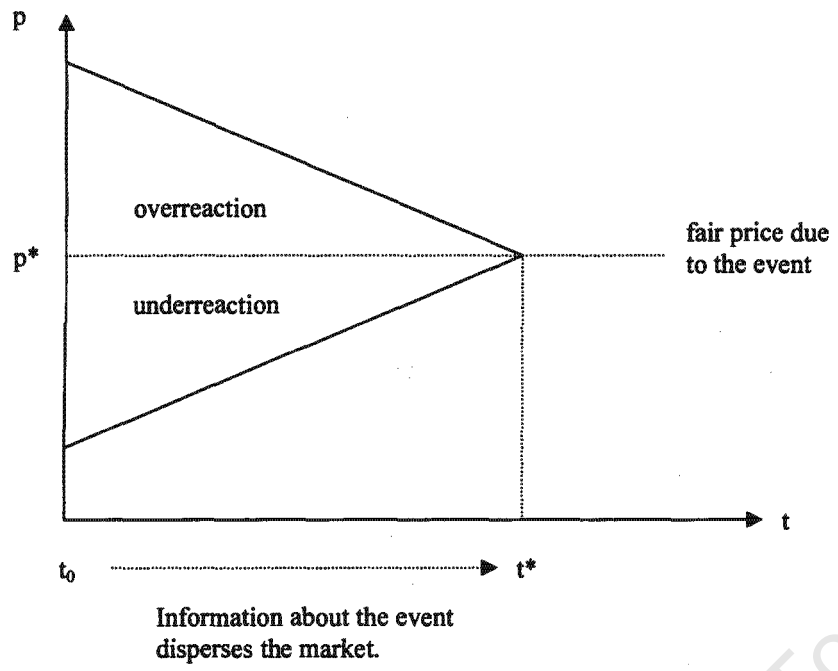


Figure 2.3 Over- and underreaction

In markets depicted by the efficient market theory we would find that $p = p^*$ at t_0. The upper and lower triangle in the graphic would not exist. In reality however, we can observe a certain amount of time, $\Delta t = t^* - t_0$, till the fair value of a security is shown by the market. Δt can be every temporal horizon in between seconds and years. Deviation from the fair value p^* is due to slowly dispersing information about an event. Whether market agents over- or underreact depends on how they interpret incoming information and how they are influenced by the factors discussed above. In other words it depends on how they conduct the formation of expectations.

Overreaction of market participants can be interpreted as a too positive assessment of a security's value. Past experience could have been positive or retrospectively perceived more positive than it actually was. In addition, less caution is being taken in interpreting new information. This behaviour can be referred to a good state of mood, to a being overconfident or simply to the desire of safety and so going with the mass. But as more information about the event disperses the market the sentiment begins to shift and the market price converges to the fair value.

Underreaction on the other hand can be regarded as an overcareful and conservative interpretation of news. As time passes though, more market

agents get a positive sentiment and the market price converges rising to the fair value.

But how can we fit price changes in periods with no news (e.g. weekends, holidays or periods of information gaps) to investors behaviour? No news is often treated as bad news. No news means that expectations about the development of investments – which are positive otherwise there would not be any investment – are disappointed. This disappointment also spreads slowly depending on the frustration tolerance or restraints of investors and makes them to withdraw from their holdings. Evidence for that is given in an article in the Business Times Money from June 4th 2006. The article "The anatomy of a crash" describes the plummeting of the South African Stock Index (JSE) despite of stable fundamentals and monetary policy. This stationary situation can therefore be interpreted as a period of no news. Nevertheless the JSE declined and Victor Hugo, CEO of Vega Asset Management labels any potential sell-off that might occur in the future rather to be anxiety driven than event driven.

Anyhow, we observe price adjustments with lag Δt which can be attributed to the behaviour of market participants. It is triggered by their expectations which are rather psychological than mathematical, i.e. rational. The lagged adjustment of security prices to information is the explanation why markets are temporarily inefficient. How long a state of inefficiency lasts is difficult to predict. It can range from within-day-reactions to within-months/years-reactions [Wär02, p. 157].

A market does not display the behaviour of a single market agent but of an aggregation of all participants. Plummer points out that there is no way to predict the behaviour of a single person. An aggregation however, is predictable since it tends to create patterns of behaviour [Plu03, p.51]. These patterns can be observed in the market as already proven. If patterns are obvious, a human being is likely to detect them without sophisticated help. But if this is the case other market agents are likely to detect the same pattern as well so that the exploitation of the pattern remains questionable. More subtle patterns on the other hand are less likely to be detected by "simple" device. This is where neural networks come into play. They can give

considerable support to an analyst in detecting and exploiting even slightest patterns.

Before we go on to the explanation of neural networks and their application in South African financial markets there should be a last remark on investment risk. The risk an analyst takes on is that his predictions are wrong. Some markets are considered as more volatile than others, i.e. less predictable and so more risky to invest in. The decisive element which creates market risk is generally identified as the rate of volatility in the market. However, since volatility is a parameter of market models it can also be argued that the actual risk lays in the choice of the model itself. If we choose a model which proves to be reliable even under highly volatile conditions, markets will loose much of their hazard and thus become accessible for rational investments. Neural networks provide the functional preconditions for reliably modelling volatility as will be shown in the following chapters.

2.4 The Black Scholes market model

A very important market segment which also allows the application of neural networks is the option market. The following section explains the market model of Black and Scholes which is widely used for pricing derivatives. In particular, light will be shed on the measure theoretic derivation of this model.

2.4.1 Introduction to the model

Fischer Black and Myron Scholes developed their model of pricing options in 1973 [BS73]. An option is a right, not an obligation to buy (call) or sell (put) an underlying asset at a predefined price at expiry (European) or even before expiry (American). Following diagrams show the payoff profiles of call and put options where K denotes the strike, i.e. the predefined buying or selling price.

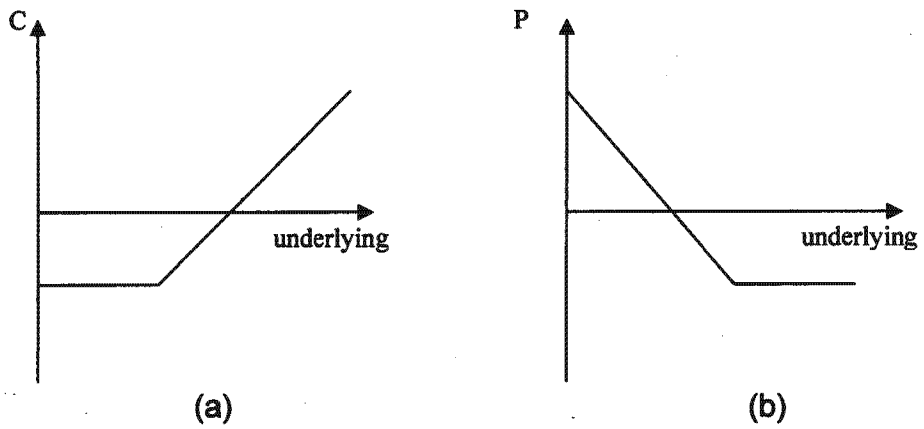


Figure 2.4 Long option payoffs: (a) call payoff, (b) put payoff

It is evident that a call option increases in value as the underlying asset price rises. Put options on the other hand gain value when the underlying asset value plummets. Given this, Black and Scholes suggested formulae for the evaluation of options [Hul03, p.246].

$$C = S(t)\, N(d_1) - e^{-r\,t}\, K\, N(d_2), \qquad (2.2)$$

$$P = e^{-r\,t}\, K\, N(-d_2) - S(t)\, N(-d_1) \qquad (2.3)$$

$$d_1 = [\ln(S/K) + (r + \sigma^2/2)\, t]/\sigma\sqrt{t} \quad \text{and} \qquad (2.4)$$

$$d_2 = d_1 - \sigma\sqrt{t}\,. \qquad (2.5)$$

2.4.2 Measure theoretic derivation

The following derivation of the option pricing formulae is oriented on a paper by Geman, El Karoui and Rochet. In their research, the authors develop a formula for the change of numeraire and apply it to options. An equally important assumption made is, that in the absence of arbitrage there is a probability measure Q such as the current price of any security is equal to the Q-expectation of its discounted future payments [GER95, p.443] as shown by the following graphic.

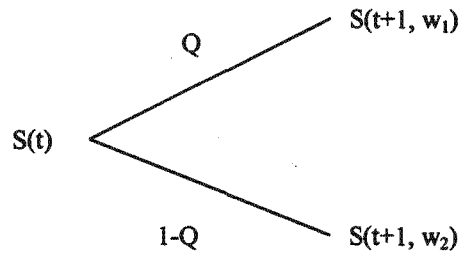


Figure 2.5 Q-martingale process

The discounted price of a security is a Q-martingale where Q is unique [Fri06, p.58]. In addition we define a probability measure Q_x relative to a numeraire X as a Q_x-martingale.
Now, let

$$V(t) = \Sigma^{n}_{k=1} w_k(t) S(t) \qquad (2.6)$$

with $V(t) \geq 0$ for all t,

be a self financing portfolio where $(w_1(t))_{t\geq0},\ldots,(w_n(t))_{t\geq0}$ are adapted, i.e. $w_1(t),\ldots,w_n(t)$ are chosen according to information available at t and $(w_1(t),\ldots,w_n(t))_{t\geq0}$ is a portfolio strategy. Then it can be defined,

$$dV(t) = \Sigma^{n}_{k=1} w_k(t)\, dS_k(t) \qquad (2.7)$$

if the stochastic integral $\int^{T}_{0} \Sigma^{n}_{k=1} w_k(t)\, dS_k(t)$ exists.

The numeraire is now a price process X(t) which is almost strictly positive for each t ϵ [0,T]. Also after changing the numeraire, a self financing portfolio remains self financing.

Let X be a new numeraire, then by Itô's formula,

$$d(S_k(t)/X(t)) = S_k(t)\, d(1/X(t)) + 1/X(t)\, dS_k(t) + d\langle S_k, 1/X\rangle_t \qquad (2.8)$$

where the last term stands for the instantaneous covariance between the semimartingales S_k and 1/X. Following this, we write,

$$d(V(t)/X(t)) = V(t)\, d(1/X(t)) + 1/X(t)\, dV(t) + d\langle V, 1/X\rangle_t \qquad (2.9)$$

for the portfolio.

Now given the self financing conditions

$$dV(t) = \Sigma_n^{k=1} w_k(t)\, dS_k(t) \qquad (2.10)$$

and

$$V(t) = \Sigma_n^{k=1} w_k(t)\, S_k(t) \qquad (2.11)$$

We come to

$$d(V(t)/X(t)) = \Sigma_n^{k=1} w_k(t) \{S_k(t)\, d(1/X(t)) + 1/X(t)\, dS_k(1) + d\langle S_k, 1/X\rangle_t\}$$

$$= \Sigma_n^{k=1} w_k(t)\, d(S_k(t)/X(t)). \qquad (2.12)$$

Comparing (2.10) and (2.12), it becomes evident, that the portfolio expressed in terms of the new numeraire remains self financing [GER95, p.446].
It follows that a contingent claim H(T) is attainable if there is a self financing portfolio with terminal value H(T).

Now let us assume there is a non-dividend paying asset n(t) and a probability π equal to an initial probability P such that for any basic security S_k without payments, the price of S_k is relative to n(t), i.e. a local martingale with respect to π.

Let us further assume that n(0) = 1, then follows

$$V(t)/n(t) \geq E_{\pi}[V(t)/n(t)|\ \mathcal{F}_t] \text{ almost surely} \qquad (2.13)$$

i.e. portfolios are super-martingales.
Is the terminal value V(t)/n(t) square integrable, the portfolio value is a π-martingale and

$$V(t)/n(t) = E_{\pi}[V(t)/n(t)|\ \mathcal{F}_t]. \qquad (2.14)$$

Now, is there a replicating portfolio whose value is always equal to the expectation, we have found the fair price of the contingent claim.
If there is another probability π^* such that

$$E_{\pi^*}[H(t)/n(t)|\ \mathcal{F}_t] = E_{\pi}[H(t)/n(t)|\ \mathcal{F}_t], \qquad (2.15)$$

we can conclude that the fair price of a contingent claim does not depend on the choice of the risk-neutral probability measure π [GER95, p.447].
Given two securities B and D, the general formula for the change of numeraire can be written as

$$B(0)\ E_{QB}[D(T)\ \phi|\ \mathcal{F}_t] = D(0)\ E_{QD}[B(T)\ \phi|\ \mathcal{F}_t] \qquad (2.16)$$

Where ϕ is any random cash flow $\mathcal{F}$ measurable. Examples of numeraires are money market accounts or zero coupon bonds [GER95, p.449].

The developed model respective its implications can now be applied to the derivation of formulae (2.2) to (2.5).

2.4.3 Black-Scholes in a measure theoretic context

We suppose that all options are attainable. Given (2.14) and (2.15), a call option formula is defined by

$$C(0)/B(0,T) = E_{QT}[(S(T)/B(t,T) - K)^{+}], \quad (2.17)$$

where K denoting the strike.
Alternatively (2.17) can be written as

$$C(0) = S(0)\, Q_S(A) - K\, B(0,T)\, Q_T(A) \quad (2.18)$$

where $A = \{w|\ S(T,w) > K\, B(T,T)\}$.

If we now allow interest rates and volatilities to be stochastic we get the forward price of S as

$$F^s(t) = S(t)/B(t,T) \quad (2.19)$$

which is a positive martingale under Q_T and can therefore be written as a stochastic integral of a Brownian process,

$$dF^s(t)/F^s(t) = \sigma_{Fs}(t)\, dW_t^{Fs}, \quad (2.20)$$

where

$$(\sigma_{Fs})^2 = (1/dt)\, Var(dF^2/F^s). \quad (2.21)$$

Assuming σ_{Fs} is deterministic and introducing Gaussian variables we conclude that $Q_T(A) = N(d_2)$ with

$$d_2 = (\sigma_{Fs}\sqrt{T})\ \{\ln(S(0)/K\, B(0,T) - 0.5\, \sigma_{Fs}\, T\} \quad (2.22)$$

Considering (2.17) we can state that

$$B(t,T)/S(t) = 1/F^{s}(t) = Z^{T}(t). \qquad (2.23)$$

Assuming that volatilities of $F^{s}(t)$ and $Z^{T}(t)$ are deterministic we write

$$Qs(A) = Qs(\sigma_{Fs} W_t - \sigma_{Fs}/2\ T \leq \ln 1/KZ(0))$$

$$= N[1/\sigma_{Fs} \sqrt{T} \ln(S(0)/KB(0,T) + 0.5\ \sigma_{Fs} \sqrt{T}\ ]$$

$$=N(d1) \qquad (2.24)$$

Defining $B(0,T) = e^{-r\,T}$, formulae (2.2) to (2.5) are finally derived in a measure theoretic context.

3. The neural network approach

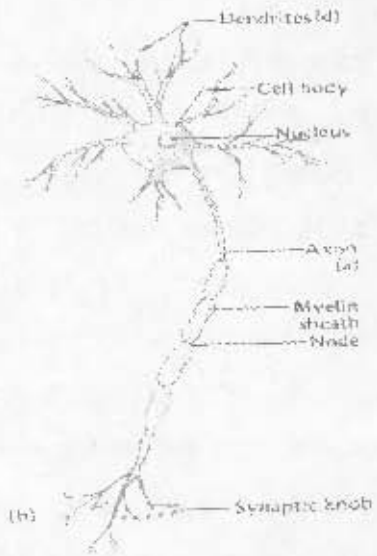
So far a vague idea was given what neural networks are, which strengths they have and that they can be used to reliably determine future security prices. Yet it has not been clarified which train of thoughts led to the development of neural networks. Most people associate the connotation "neuron" with cells in our brain rather than with an electronic decision system so chapter 3.1 depicts the neuron in its biological sense. But exactly this communality makes this particular decision system so unique. Neural networks are constructed like a brain and imitate its functionality. The transformation from biology to a mathematical system (i.e. an artificial neuron) will be subject of chapter 3.2. The transfer function as an important integral element of an artificial neuron is introduced in chapter 3.3. In order to generate decent results, a model has to reflect all particularities of its field of application. Again, it might not be completely obvious what the biological brain – as the underlying concept – has in common with markets. This question will be answered in chapter 3.4.

3.1. The biological neuron

The nervous system provides rapid communication between the various tissues and organs of the body [She83, p.327]. To effect rapid reaction to stimuli, quick and direct pathways for conducting messages are required [She83, p.328]. Nervous cells, or neurons, guarantee the fast transfer of stimuli. They are specialised to conduct electrochemical information at high speed and bundled together, are able to regulate the direction of the information flow. Masses of nervous cells form control centres such as the brain which acts as a central clearing house for information and point of coordination. Thus the neuron becomes the basic unit of the nervous system [She83, p.328].

A neuron consists of three parts [She83, pp.328] [DN91, pp.3]. The *cell body* contains the nucleus and carries out the biochemical transformations necessary to the life of a neuron. The cell body is surrounded by a hair-like structure called dendrites [DN91, p.3]. *Dendrites* receive messages from other cells [She83, p.329] [DN91, p.4]. The *axon* as the third part of the neuron passes information to connected neurons [She83, p.329]. The language with which neurons communicate is an electrochemical one. Any functions of the human mind are carried out by neurons via electrochemical changes called impulses or stimuli. Stimuli cause the membrane of a neuron to depolarise shortly. A depolarised membrane allows the exchange of chemicals. The rapid depolarization-repolarization is called the action potential [She83, p.331]. The minimal strength of a stimulus to generate an action potential is the threshold stimulus. The cell body carries out the summation of all incoming impulses [DN91, p.5]. Are the aggregated stimuli too weak, the neuron will not react [She83, p.333] [Ami92, pp. 12]. An illustration of such a neuron is given on the subsequent page.

Each neuron has a large quantity of synapses. Learning is achieved by changing the effectiveness of the synapses so that the influence of one neuron on another one changes [SS96, p.4].



Graphic 3.1 The biological neuron [Sho80]

3.2 The artificial neuron

In order to achieve human-like performance from computers, artificial neurons have been developed. Artificial neurons encapsulate what is known about the biological neurons in the human brain. Artificial neurons can be connected to tight networks. The network learns by modifying these connections based on same external stimulus [Zir97, p.3]. Figure 3.1 illustrates an artificial neuron, i.e. the mathematical version of the biological neuron.

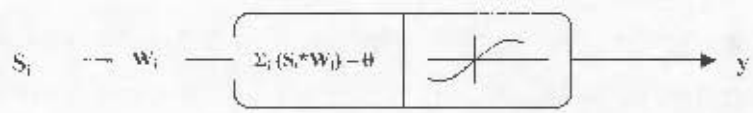


Figure 3.1 The artificial neuron

Any incoming signal S_i is linked to the neuron body through synapses. Weights w_i are attributed to each synapse. These weights can be either positive or negative [Azo94, p.12]. By changing the weights, signals obtain relative importance. Inside the neuron, the sum of incoming signals, respective their weights, is taken and modified by a bias 0. The result is

called the net input [BDHO5, p. 2-2] and is passed through a function which is called non-linearity, activation or transfer function [Zir97, p. 4]. Is the result of this non-linear transformation larger than the threshold value of the respective neuron, an output y is generated [DN91, p. 19].

The bias – often with value 1 – feeds into all neurons of the network. It explicitly represents the levels of the adjustable neurons in the transfer function input. Thus it eliminates the need to treat thresholds as specific neuron features and grants a more efficient implementation of algorithms [Azo94, p.14]. The way artificial neurons learn will be explicitly introduced in Chapter 4. In connecting each artificial neuron to one another, we create a network of artificial neurons. The output of such a network depends on the output of each neuron and on the order in which they are connected. A network as described resembles the biological brain not only in its structure but also in its functionality with respect to signal processing and learning.

3.3 The transfer function

As already mentioned, one advantage of neural networks is the non-linear processing of data. It was pointed out in chapter 3.2 that this is achieved by passing the net input through a non-linear function, known as activation or transfer function. In other words, a transfer function is the rule for mapping the neuron's summed input to its output and so introducing non-linearity into the network design [Azo94, p.52]. Zirilli distinguishes three general types of non-linearities which are referred to as hard-limiter, threshold-logic or piecewise linear and sigmoid non-linearity [Zir97, p.5]. A more detailed list of transfer functions is given by Azoff [Azo94, pp.51]. According to Zimmermann, sigmoid functions are the most common ones used in artificial neural networks [Zim94, p.5]. In the following, some important transfer functions shall be explained in more detail [Azo94, pp.52].

- The *hard-limiter* function has the following form,

$$\Phi(x) = \begin{cases} 1 \text{ for } x \geq 0 \\ 0 \text{ for } x < 0. \end{cases} \qquad (3.1)$$

Hard-limiter functions are suitable for binary class problems that take one of two values. The problem however, is that it is not differentiable and so not suitable for the backpropagation method as will be shown in chapter 4.

- The *threshold-logic* or *piecewise linear* function has

$$\Phi(x) = \begin{cases} 1 \text{ for } x \geq z \\ x \text{ for } -z < x < z \\ 0 \text{ for } x \leq -z. \end{cases} \qquad \text{with } z \in \mathbb{R} \qquad (3.2)$$

Threshold-logic functions incorporate the same problem as the hard-limiter functions do. They are indifferentiable in the points –z, z and so not suitable for backpropagation.

- A commonly implemented function is the *sigmoid function*. It passes the input value of the range $x \in [-\infty, +\infty]$ into an output range $\Phi(x) \in [0,1]$. The sigmoid function is represented by

$$\Phi(x) = (1 + \exp(-x\, g))^{-1}, \qquad (3.3)$$

where g determines the slope. Sigmoid functions are smooth and can so be differentiated at any point. This property makes them suitable for backpropagation.

- The *tanh* function squashes any input $x \in [-\infty, +\infty]$ into an output $\Phi(x) \in [-1,1]$ and has the form

$$\Phi(x) = 2 / (\exp(-2\, x) + 1) - 1. \qquad (3.4)$$

Tanh is differentiable anywhere and is one of the most common used transfer functions.

- The *Gaussian* function has the form

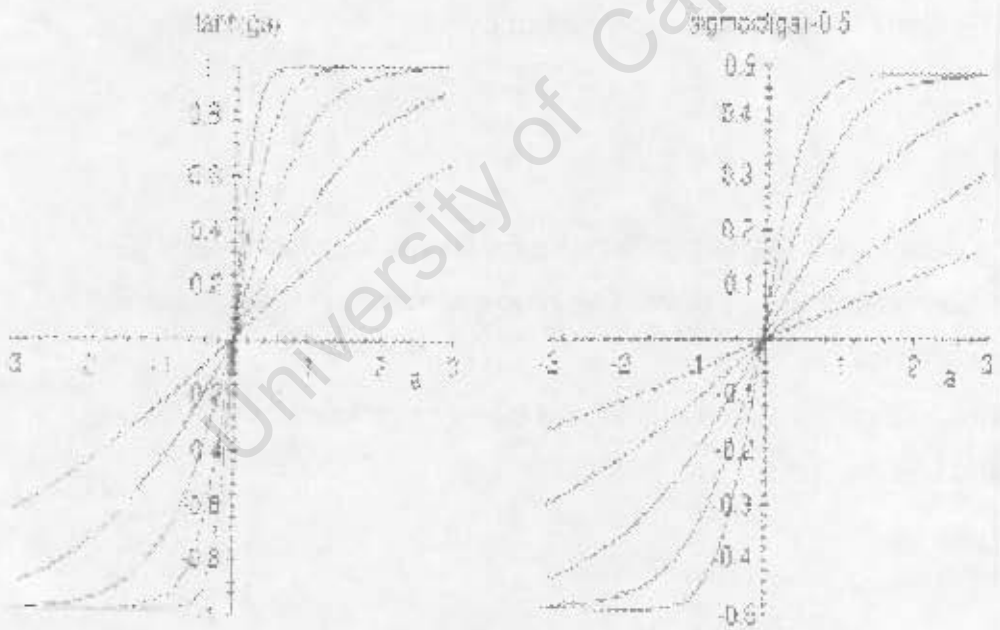
$$\Phi(x) = 1 / \sigma \sqrt{2 \pi} \exp(-(x - \mu)^2 / 2 \sigma^2 \quad (3.5)$$

for the one dimensional and

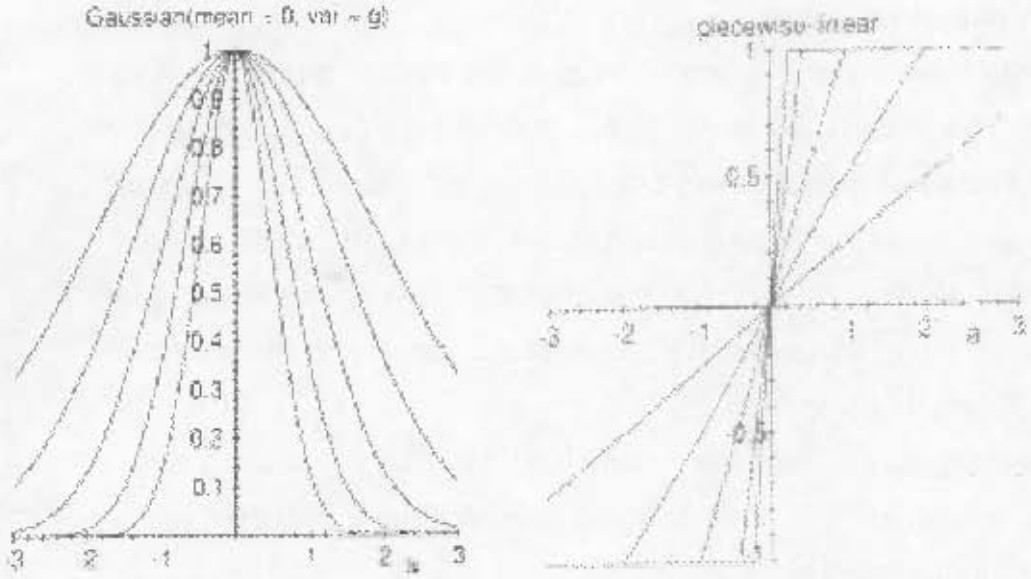
$$\Phi(x) = \exp(-0.5 (x - \mu)' \Sigma^{-1} (x - \mu)) / \sqrt{(2\pi)^n |\Sigma|} \quad (3.6)$$

for the multivariate case. Gaussian functions are used in radial basis functions (RBF), a special case of feedforward neural networks introduced in chapter 4.

Graphic 3.2 illustrates each of the introduced transfer functions.



(a) hyperbolic tangent function (b) sigmoidal function



(c) Gaussian density function (d) piecewise-linear function

Graphic 3.2 Common transfer functions [Azo94]

3.4 Analogy of biology and markets

Just like the brain, markets are considerably complex systems. In a market all participants are connected to each other. Apart from face-to-face communication, market agents use phones, faxsmiles or any kind of electronic device to exchange information. These agents, e.g. an equity trader, represent the biological neurons. Just like the neuron is involved in the exchange of impulses, a trader provides and receives information. Since the amount of information is generally vast, it has to be filtered with respect to importance. In the biological context, filtering information is similar to the effectiveness of synapses. After all relevant information is collected, it has to be aggregated. A trader would edit and structure his data to draw reasonable conclusions. In the same way impulses are assembled and summed up in the neuron body. Eventually the market agent has to decide on basis of his information whether or not to take action. A decision to act will only be taken if the quality of information eradicates any doubt. This is comparable to the threshold of a neuron [Zim05, p.5].

If we assume that a single neuron reflects the decision making scheme of a single trader, a neural network with a large quantity of nods depicts a market [Gro02, p.23].

4 Training of neural networks

Like the human brain, neural networks have to be trained in order to meet their purpose. This is achieved by iteratively changing the free parameters of the network, known as weights and bias [Azo94, p.3]. Literature generally describes three ways of training a neural network. If the output of the network is known during training, the training method is referred to as *supervised learning*. The error, i.e. the difference between the actual and the expected output is fed back into the network.

Reinforced learning as a second training method does not provide any target value. The only criterion for a feedback into the algorithm is, whether or not the network performance is correct.

Unsupervised learning eventually does not know any feedback. Learning is solely achieved by self-organising in using correlations among input data [Azo94, p.4] [BDH05, p.4-3].

The following chapters will exclusively concentrate on supervised learning. In chapter 4.1 an error function is introduced as the subject of the optimisation problem. This optimisation however, is problematic in multi-layer networks. The remedy for this problem is the backpropagation algorithm which is subject of chapter 4.2. Chapter 4.3 introduces three common learning rules and chapter 4.4 the concept of pattern recognition. Since inputs are approximated by neural networks, chapter 4.5 introduces an approximation algorithm.

4.1 The optimization problem

Given supervised training, the rational for the use of neural networks is predicting a given target variable y from information on a set of observed input variables x [McnC5, p.13].

The goal of the optimization process is it to minimize the error, i.e. the deviation of the output to the target. This consideration leads to the following error function E,

$$E = 1/T \sum_{t=1}^{T} E_t (NN(x_t, w) - y_t^{target})^2, \qquad (4.1)$$

where $NN(x_t, w)$ represents the weighted and nonlinearly transformed input and y_t^{target} is the target used for supervised training [Zim05, p.15]. Since the deviation between the network output and the target is the subject of minimization, the distance must be a squared difference. The overall error is averaged out over the total number of input-target pairs T. This error function is also called the mean square error (MSE) [Gro02, p.115]. As pointed out in McNelis, there exists a considerable amount of different types of error functions [Mcn05, pp.13].

The advantage of the MSE approach compared to linear regression models is that by applying nonlinear transformations, highly volatile conditions can be reliably simulated [Mcn05, p.15].

Compared to the nonlinear GARCH approach, the respective MSE function is not limited to a well defined number of parameters or specific distributions [Mcn05, p.17].

4.2 The backpropagation algorithm

Supervised training of multi-layer neural networks turns out to be somewhat delicate since information about the net error is not directly available for the hidden layers [Gro02, p.113]. The remedy to this problem is the backpropagation algorithm which is a generalization of the Widrow-Hoff rule for multi-layer perceptrons developed by Rummelhart, Hinton and Williams. The central idea is to use smooth, differentiable transfer functions [DN91, p.47] and to derive the first partial derivatives of the neural network error function with respect to the network parameters [Zim94, p.38]. Given this, learning rules optimize the network by adjusting the weights such that the difference between the outputs and the associated observations are minimized on average [Zim94, p.40].

In the following we explain the standard backpropagation algorithm [BDH05, pp.11-1] on the basis of a multi-layer feedforward neural network as depicted by Zimmermann [Zim94, p.37]. The network comprises three layers. One input layer with $i = 1,..., I$ neurons, one hidden layer with $j = 1,...., m$ neurons and one output layer with $k = 1,...., n$ neurons. Concerning simplicity, we decline on the use of a bias. The flow of information inside of the network is according to the direction of the arrows as illustrated in figure 4.1.

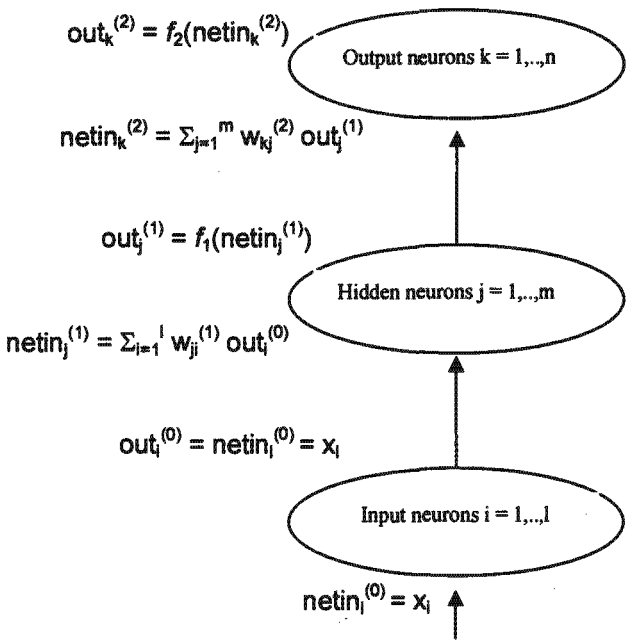


Figure 4.1 Flow and transformation of information

The input x_i is passed forward through the network. The input layer does not apply any weight matrix or transfer function so that the net input $input_i$ equals the net output out_0. At the hidden layer and output layer however, weights and transfer functions are provided so that the input finally results in $out_k^{(2)}$. This output is compared to the target values and the deviation is computed. At this point the MSE function derived in chapter 4.1 comes into play. In order to minimize the deviation, the weights $w^{(0)}$ and $w^{(1)}$ have to be adjusted such that

$$MSE = E_t = 1/T \sum_{t=1}^{T} \sum_{k=1}^{n} \tfrac{1}{2} (out_k^{(2)} - target_k)^2 \rightarrow \min_{w_{ji}^{(0)},\, w_{kj}^{(1)}}! \qquad (4.2)$$

The error is backpropagated through the layers and the residual error added to the new input [Azo94, p.17]. The so modified input is again passed through the net - while weights are changed - compared to the target and the error is backpropagated. This procedure is repeated until the residual error is minimal, i.e. the MSE function has arrived at its global minimum. In the hidden layers however, the error is an indirect function of the weights. That is why we have to apply the chain rule of calculus in order to calculate the derivatives [BDH05, p.11-9].

The first application is to find the partial derivatives of the MSE function with respect to $w_{kj}^{(2)}$. This is the weight which is located between the output and the hidden layer.

$$\partial E_t / \partial w_{kj}^{(2)} = 1/T \sum_{t=1}^{T} (out_k^{(2)} - target_k) f_2(netin_k^{(2)}) out_j^{(1)} \quad \text{with} \qquad (4.3)$$

$$\delta_k^{(2)} = (out_k^{(2)} - target_k) f_2'(netin_k^{(2)}), \qquad (4.4)$$

where $(out_k^{(2)} - target_k)$ is the network error and $f_2'(netin_k^{(2)})$ the partial derivative with respect to $w_{kj}^{(2)}$. Substituting (4.4) into (4.3) gives

$$\partial E_t / \partial w_{kj}^{(2)} = 1/T \sum_{t=1}^{T} \delta_k^{(2)} out_j^{(1)} \qquad (4.5)$$

To obtain the partial derivative of the error function with respect to the weight $w_{ji}^{(1)}$, we have to repeat the described procedure in between the input and hidden layer so that

$$\partial E_t / \partial w_{ij}^{(1)} = 1/T \sum_{t=1}^{T} \sum_{k=1}^{n} \delta_k^{(2)} w_{kj}^{(2)} f_1'(netin_j^{(1)}) out_i^{(0)} \quad \text{with} \qquad (4.6)$$

$$\delta_j^{(1)} = f_1'(netin_j^{(1)}) \sum_{k=1}^{n} w_{kj}^{(2)} \delta_k^{(2)}. \qquad (4.7)$$

Substituting (4.7) into (4.6) yields

$$\partial E_t / \partial w_{ij}^{(0)} = 1/T \sum_{t=1}^{T} \delta_j^{(1)} out_i^{(0)}. \qquad (4.8)$$

As a result we got the partial derivatives (4.5) and (4.8) with respect to their specific weights, also called (cumulative) gradients [Zim94, pp.39]. As a next step the backpropagation algorithm is to be applied. This is illustrated in the figure below.

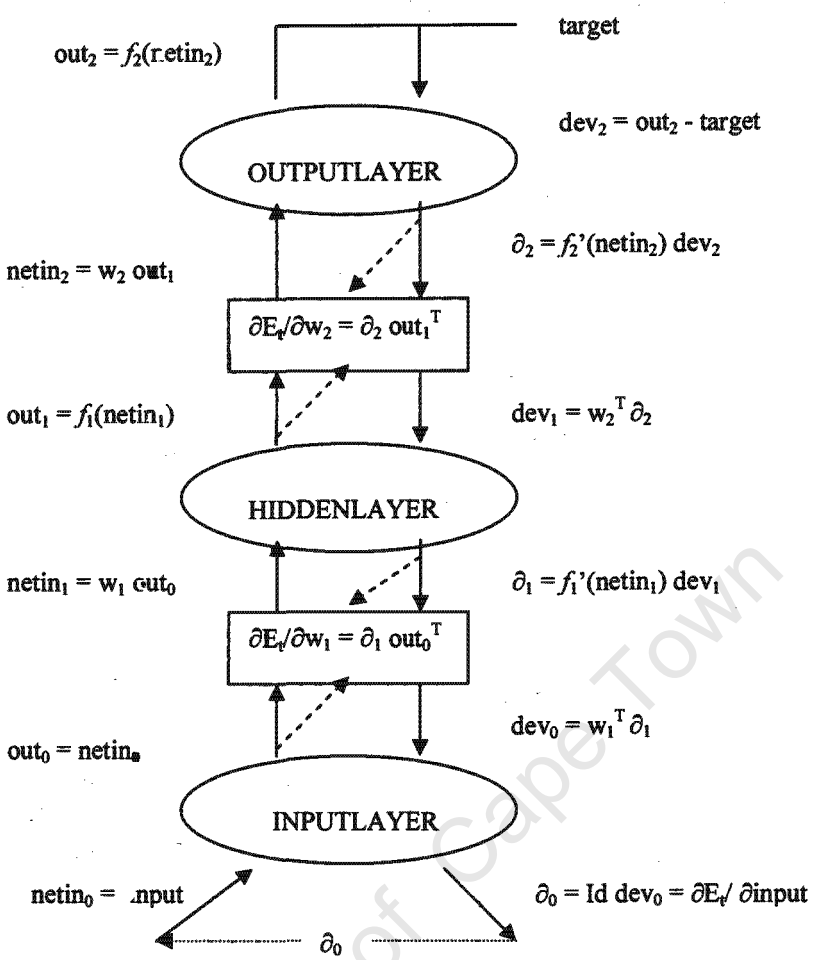


Figure 4.2 Error backpropagation. Id is the identity matrix.

The cumulative gradients establish a cross product between the forward and the backward path [Zim94, pp.39]. In particular, the output $out_j^{(0)}$ and $out_k^{(1)}$ is multiplied by the auxiliary terms $\delta_k^{(2)}$ and $\delta_j^{(1)}$ [Gro02, p.118].

Finally the error is backpropagated through the net and the residual error ∂_0 is reused as an input correction term. This algorithm is looped until equation (4.2) has reached its minimum.

4.3 Learning rules

The backpropagation algorithm provides a procedure to calculate the residual error of a multi-layer perceptron. Yet it gives no suggestion what so ever of how weights should be modified in order to attain the minimal residual error. That is why in addition to the backpropagation algorithm, learning rules are required which determine the change of weights [Zim94, p.40]. The optimization of the error function is a high dimensional problem with a high

degree of non-linearities. Under such conditions, learning is an iterative search through the solution space by adjusting the weights according to a certain learning rule [Zim94, pp.40]. The connection between the error function and the weights is illustrated in the subsequent graphic.

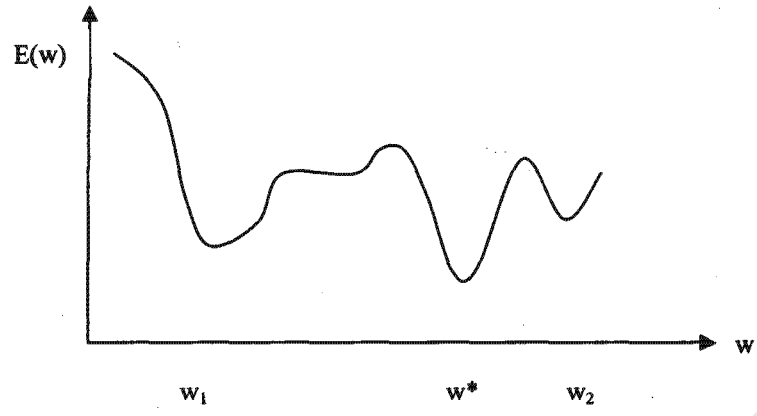


Figure 4.3 Error function and weight values

The residual error as shown, depends on the values of the weight vector w. The overall goal is it, as pointed out in chapter 4.1 to find the minimal deviation, i.e. to find the global minimum indicated by w^*. Weight vectors w_1 and w_2 only yield local minima, i.e. their solutions are sub-optimal. The potentially high amount of local minima is the reason why neural computing takes time and requires alternative approaches [Mcn05, p.65].

In the learning process weights are revised from one step to the next. This can be mathematically expressed as

$$w_{t+1} = w_t + \eta \, \Delta w_t \, . \qquad (4.9)$$

As pointed out, the modification of weights depends on the length of the learning step η and on the alternation of the respective weight Δw_t [Zim94, p.40]. η has to be sufficiently small in order to guarantee convergence in local minima [Zim94, p.41]. However, is η too small, the time to convergence becomes unreasonably long. Figure 4.4 illustrates both cases.

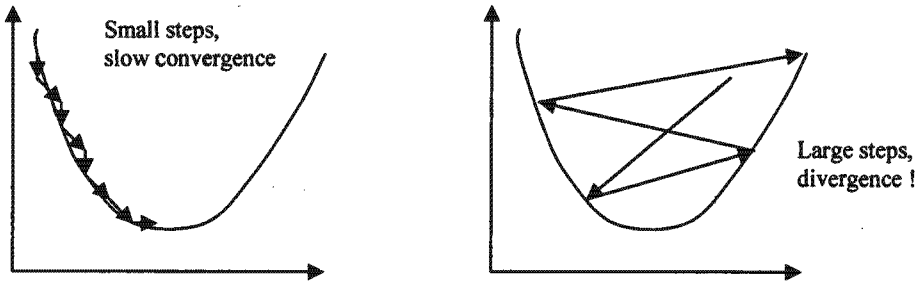


Figure 4.4 Missproportioned learning rates

In the following we explain three common learning rules. Learning rules do not only determine how weights are changed but also the search direction Δw_t.

4.3.1 Steepest descend learning

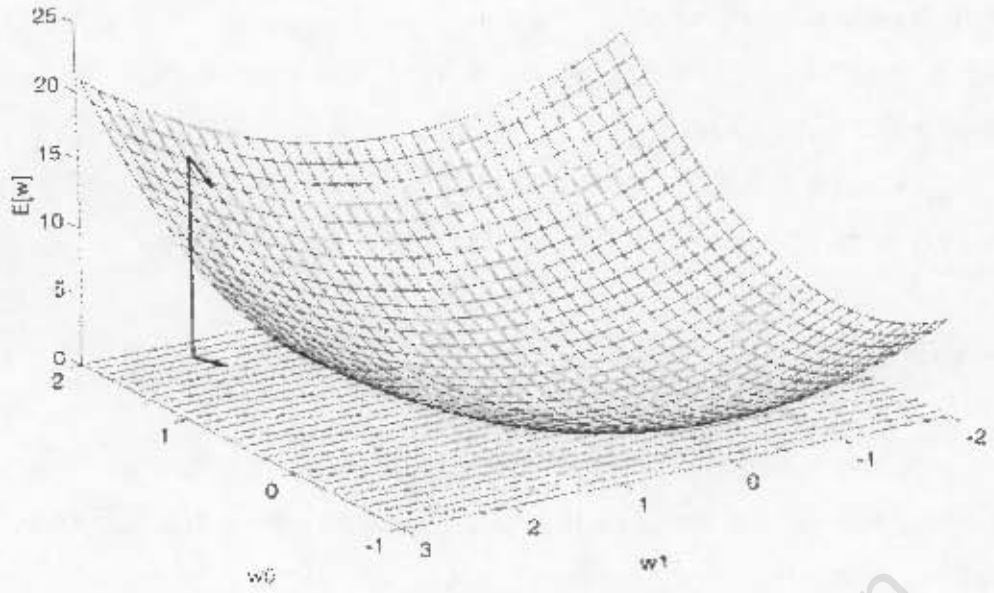
The key idea of the this method is to use the gradient descend to search the solution space for weight vectors which minimize the mean square error [Har04, pp.20]. As derived in chapter 4.2, the gradient is the partial derivative of the error function with respect to its weights and is provided by the backpropagation algorithm. Following, we can indicate the gradient vector as

$$\nabla E[w] = [\, \partial E/\partial w_0, \partial E/\partial w_1, \ldots \partial E/\partial w_n]. \qquad (4.10)$$

The negative of vector $\nabla E[w]$ specifies the direction of the steepest descend so that the learning rule can be written as

$$\Delta w = -\eta \nabla E[w]. \qquad (4.11)$$

Starting with an arbitrary initial weight vector, repeated modification in small steps transforms the vector in the direction that produces the steepest descend along the error surface [Har04, p.24]. Graphic 4.1 illustrates this.



Graphic 4.1 Direction of steepest descend [Har04]

Writing (4.11) in component form we get

$$\Delta w = -\eta\, g \qquad (4.12)$$

with,

$$g = 1/T \sum_{t=1}^{T} g_t = \nabla E[w] \qquad (4.13)$$

and

$$g_t = \partial E_t / \partial w, \qquad (4.14)$$

where g is the gradient and T the total number of runs [Gro02, p.132] [Har04, p 29] [Zim05, p.16].

The gradient descend algorithm can be exercised as long as the calculated gradient g is non-zero. However, if g = 0, the network has converged to a local minimum [Gro02. p.132].

The major problem of this method however, is that the learning progress is often stuck in sub-optimal local minima since the detection of inefficiencies is not totally sophisticated [Zim94, p.41].

4.3.2 Pattern-By-Pattern learning

Following this method, weights are adjusted after the evaluation of each training sample [Gro02, p.133].

For each single training pattern t, the search direction Δw_t of pattern-by-pattern learning is

$$\Delta w_t = -g_t = \partial E_t/\partial w = -g_t\,\eta = -\eta\, g - \eta(g_t - g). \qquad (4.15)$$

$-\eta\, g$ represents the steepest descend learning. The term $\eta(g_t - g)$ is a stochastic component which ensures that through large jumps, the learning progress will not be caught in local minima [Zim05, p.16].

4.3.3 Vario Eta learning

Vario Eta learning extends the learning rate η by a variable length [Zim94, pp. 48]. This results in an individual scaling of the weight changes [Gro02, p.134] [NZ98, p.396]. From equation (4.15) we can see that the learning rate η is scaled by the standard error of the squared deviations $(g_t - g)^2$ [Gro02, p.134] [NZ98, pp.396] [Zim94, pp.48]. The learning rule can be described as

$$\Delta w_t = -1/\sqrt{\Sigma_t (g_t - g)^2}\, g_t$$

$$= -1/\sqrt{\Sigma_t (\partial E_t/\partial w - \partial E/\partial w)^2}\, \partial E_t/\partial w. \qquad (4.16)$$

Equation (4.16) implies that multiplying the search direction Δw_t by the learning rate η, each weight is changed by an individual increment .

Vario Eta learning behaves like a stochastic approximation of a quasi-Newton method [Gro02, p.134] [NZ98, p.396]. This implies that Vario Eta rules damp the learning jumps and thus converge faster to the minimum [Gro02, p.134].

4.4 Pattern recognition

The following chapter explains the concept of pattern recognition and describes how neural networks process this task. Pattern Recognition is the learning of input structures by data classification, including a later identification of the learnt patterns. Yet pattern recognition has another dimension which is referred to as generalization. From a set of training data,

neural networks should be capable to correctly infer out of the sample points, i.e. values for which they have not been trained [Bra, p.9]. Neural networks proved to be ideal for this task due to certain properties such as adaptive learning, self-organization, fault and noise tolerance [Jes, pp.3].

4.4.1 Phases and methods of pattern recognition

Pattern recognition is conducted in three steps [Jes, pp.2]:

- *Filtering* is the removal of unwanted input information,
- *Feature extraction* as a process of deriving useful information from the filtered inputs and
- *Classification* where the input objects are attributed to certain categories.

A more detailed listing of the stages involved in pattern recognition are given by Davalo [DN91, pp.36].
Different methods can be distinguished when classifying inputs:

- *Member-roster* concept where similar patterns are stored in a classification system. As soon as an unknown pattern appears, a new container is opened,
- *Common property* concepts group patterns according to certain attributes. This is somewhat more general than the first method. A new class of patterns is generated if a pattern does not posses attributes inherent to existing classes,
- *Clustering* concepts represent patterns in vectors which components $c_i \in \mathbb{R}$. By measuring the distance between new patterns and already classified patterns, the membership of the new pattern to the existing classes is tested. Is the distance minimal to a certain class, the pattern will be classified under it.

The following hierarchy illustrates the phases and methods of pattern recognition.

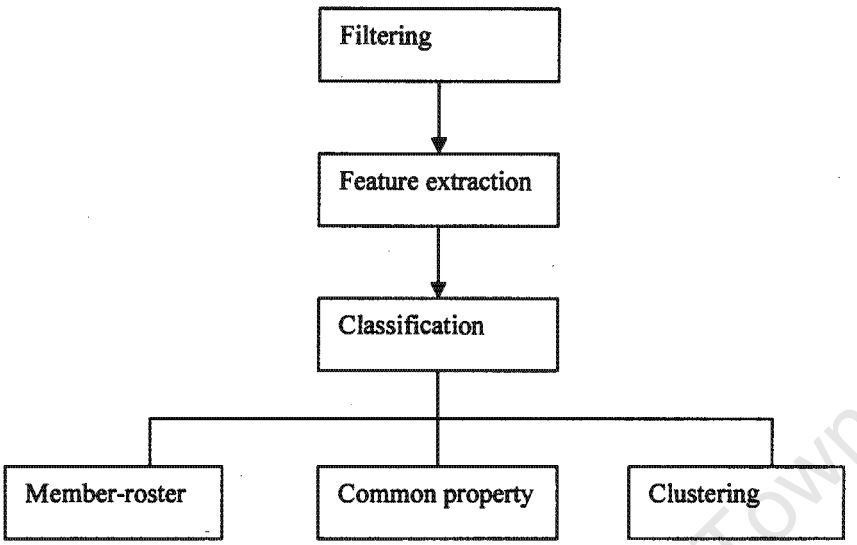


Figure 4.5 Phases of pattern recognition

After having introduced the general concepts behind pattern recognition, the next chapter will show how a neuron classifies input data.

4.4.2 The classification algorithm

A single neuron can classify data into two categories. A very central concept is the decision boundary. This is a hyperplane which crosses the output space and divides it into two classes. The following output function with two inputs shall be used as an example.

$$y = \text{hardlims(net input)} = \text{hardlims}[(w_{11},w_{12})\ (x_1\ x_2)^T + b] \qquad (4.17)$$

where w_i are the weights, x_i the inputs and b the bias. A decision boundary is characterized by y = 0. Is the inner product of weights and inputs larger than –b, the output is 1. If less than –b, the output is -1. Figure 4.6 illustrates a classification example of many inputs where the squares have function values 1 and circles -1.

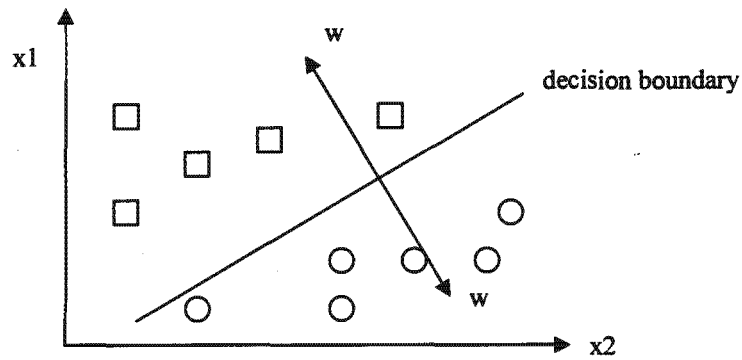


Figure 4.6 Data classification

If the decision boundary is linear as indicated in the graphic, patterns can only be recognized if inputs are linearly separable. Weight vectors are poised orthogonal to the boundary. By changing these vectors, the boundary can be adjusted among the data points. How boundaries change depends on the learning rules introduced in chapter 4.3. The change occurs according to a minimal error function.

For networks with multiple neurons, we get decision boundaries for each neuron. Following graphic shows the decision space for three and four neurons.

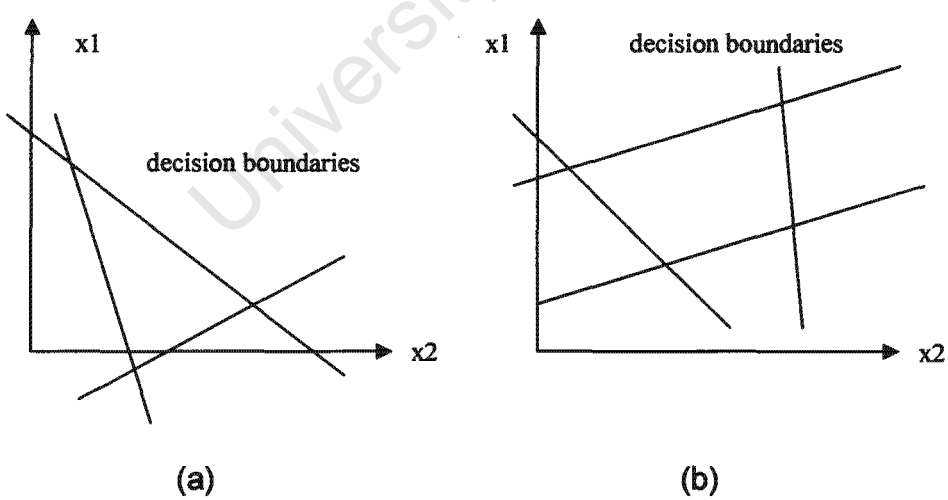
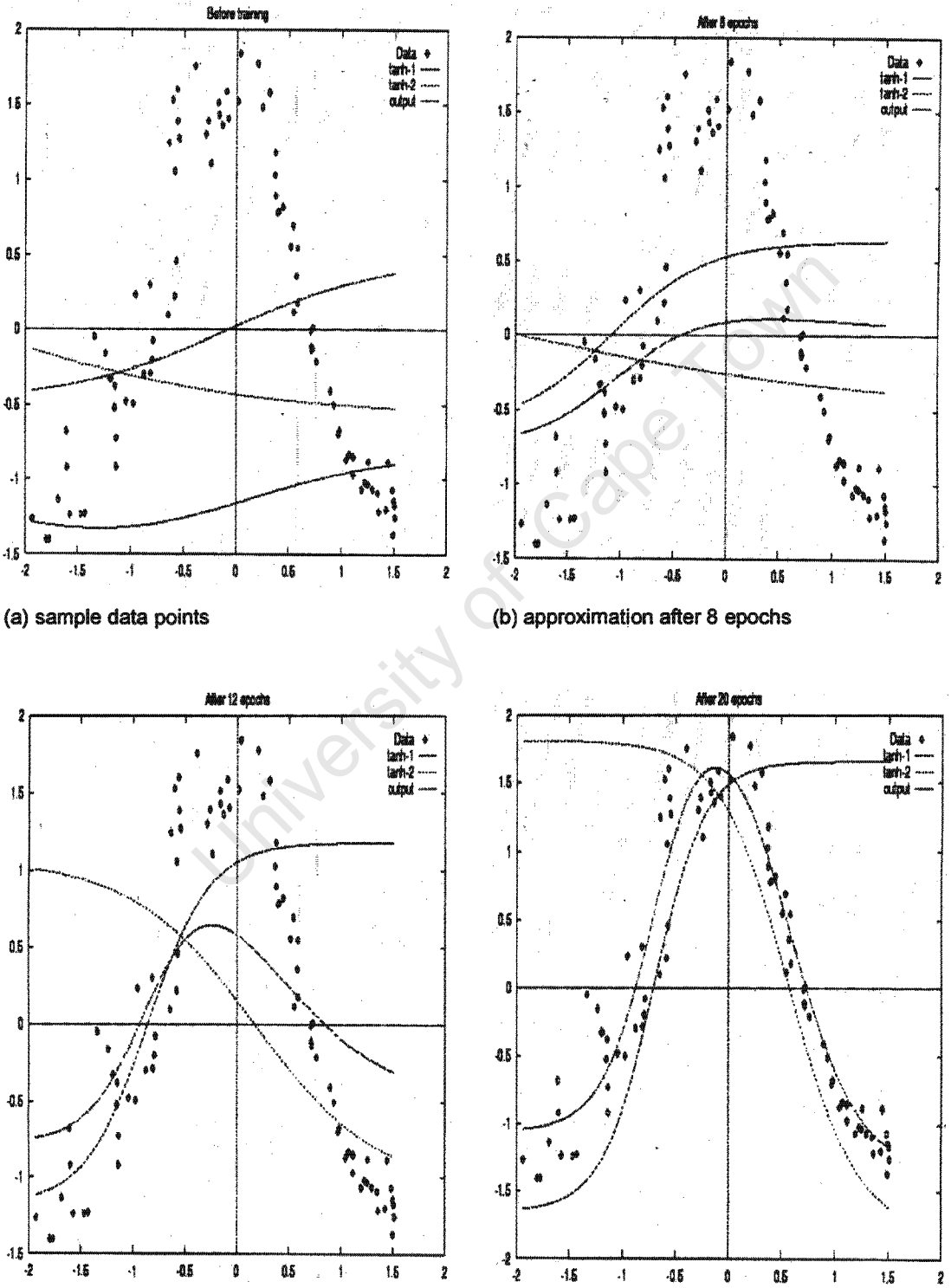


Figure 4.7 Decision spaces for (a) three and (b) four neuron networks

Many cases, forecasting time series in particular, require non-linear decision boundaries. An example could be

$$y = \tanh(\text{net input}) = \tanh[(w_{11}, w_{12})\,(x_1\ x_2)^T + b]. \qquad (4.18)$$

Graphic 4.2 illustrates that by adjusting weights, the boundary function gradually minimizes the distance to the data points.



(a) sample data points (b) approximation after 8 epochs

(c) approximation after 12 epochs (d) approximation after 20 epochs

Graphic 4.2 Non-linear approximation of data [SS96]

The fourth image shows the best approximation after 20 epochs. The approximated patterns are stored in container classes as introduced in the previous chapter. Now, if the network recognizes a sequence of patterns, it can infer an out of the sample guess by choosing the right pattern from the storage.

4.5 Approximation theory

The last chapter gave a general idea of how neural networks conduct pattern recognition. It was shown that classification of data is a central concept by which a linear or non-linear decision boundary is plotted in the data field. The algorithm is derived from a minimization problem. The boundary is fitted into the data field such that the distance between the data points and the boundary is minimized. This optimization is attained by changing the weights of the neurons and so the series of data points can be represented by a function as illustrated in graphic 4.2.

However each optimization – especially if heuristic – has an inherent problem which is the dichotomy between accuracy and computation effort. It is logical that extremely thorough results require much computational resources.

Mathematically it is possible to approximate complex functions by more simple ones. This is the base on which neural network operate. Approximation errors are due to unprecise measurement of data or the use of approximations instead of real data. Given v_r as the real value and v_a as the approximation, we can define the error as,

$$\varepsilon = | v_r - v_a | \qquad (4.19)$$

Approximations are typically achieved using polynomials [WIK] or power series expansions in which higher order terms are dropped.

In the following Remes' algorithm shall be explained, representative for a whole variety of methods [Wat80, pp.69].

This method generates an optimal polynomial P(x) to approximate a given function f(x) over a defined interval. It iteratively converges to a polynomial which has an error function with N+2 level extrema. Remes' algorithm uses the possibility to construct an Nth degree polynomial that leads to alternating

errors, given N+2 test points. Given N+2 test points x_1, x_2,...x_n, following equations have to be solved,

$$P(x_1) - f(x_1) = +\varepsilon \quad (4.20)$$

$$P(x_2) - f(x_2) = -\varepsilon \quad (4.21)$$

$$P(x_3) - f(x_3) = +\varepsilon \quad (4.22)$$

.

.

$$P(x_n) - f(x_n) = +/- \varepsilon \quad (4.23)$$

which is,

$$P_0 + P_1x_1 + P_2x_1^2 + P_3x_1^3 + ...+ P_nx_1^n - f(x_1) = +\varepsilon \quad (4.24)$$

$$P_0 + P_1x_2 + P_2x_2^2 + P_3x_2^3 + ...+ P_nx_2^n - f(x_2) = +\varepsilon \quad (4.25)$$

etc.

All powers and functions are known, i.e. the above equations are just n+2 linear equations in the n+2 variables P_0, P_1,....,P_n and ε.

We get the polynomial and ε by simply solving the system of equations.

Apart from Remes' algorithm Szabados suggests linear interpolation as a way to obtain approximated results [Sza06]. Eventually, as graphic 4.2 shows, a tanh function was used by the network to approximate the given data pattern.

5 Feedforward neural networks

This chapter gives an overview of the architecture of feedforward neural networks. Literature provides a rich variety of different kinds of feedforward networks whereas the most important ones are referred to as the multi-layer perceptron and the radial basis function network [Tar98, p.87]. Both networks have to be trained under supervision a concept introduced in the previous chapter.

Feedforward networks allow signals to travel one way only which is from the input to the output. There is no feedback, i.e. any output of one layer does not affect the same layer anymore [SS96, p.9]. Signals are not looped inside of the network but sent straight forward. Feedforward neural networks are

preferably used for pattern recognition [SS96, p.9]. A typical feedforward neural network is illustrated in figure 5.1.

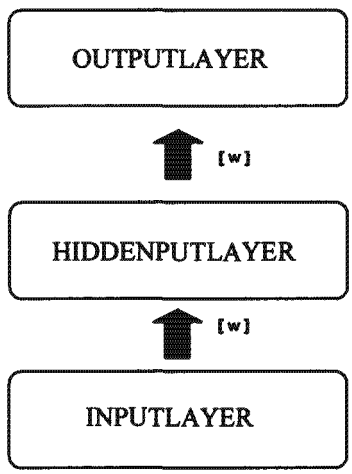


Figure 5.1 General architecture of feedforward neural networks

The graphic shows a networks consisting of three layers. The input layer introduces data to the network, the hidden layer processes them and the output layer finally states the results of the network. As indicated by the arrows, signals are propagated straight forward from the input to the output. The modelling of loops is not allowed. Weight matrices w and non-linear transformation by transfer functions transform the input to output. Again, a bias which contributes to the input transformation in each neuron is not displayed by the graphic. Chapter 5.1 and 5.2 introduce multi-layer perceptrons and radial basis networks in more detail. Special focus is being shift on the difference in the activation of neurons and on the respective output equations. Chapter 5.3 summarises the difference between multi-layer perceptrons and radial basis networks. Since model uncertainty is problematic when conducting forecasts, chapter 5.4 provides an idea of how to reduce uncertainty.

5.1 The multi layer perceptron

Multi-layer perceptrons are the most general form of feedforward neural networks. The following figure shows the architecture of a multi-layer perceptron. In order to reduce complexity, the displayed network has only three layers.

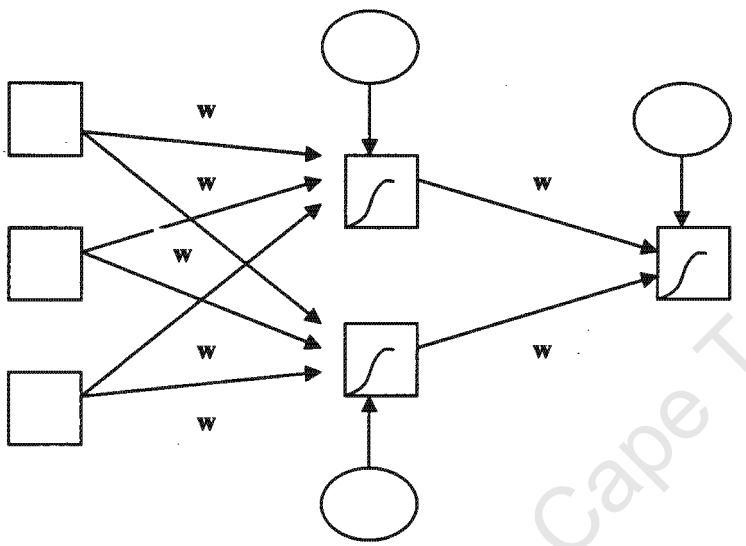


Figure 5.2 The architecture of a multi-layer perceptron

Theoretically a network can be endowed with infinitely many layers but typically has only one or two [Azo94, p.13]. Again, for simplicity let us assume that the network posses three neurons in the input layer, two in the hidden and one in the output layer. Attached to each flow of information is a specific weight w which effects linear transformations of input values. The oval elements of the graphic stand for the biases which are integrated parts of each neuron.

This illustrated network with one hidden layer is the most basic and commonly used structure in economic and financial applications [Mcn05, p.22]. Multi-layer networks usually use a sigmoidal or the tanh function for non-linear transformation in the neurons [Zim05, p.26]. Generally, a network design may deploy a different transfer function in each layer or neuron [Azo94, p.13].

Given the network's structure and the transfer function an output equation ca easily be derived. We write

$$n_{k,t} = w_{k,0} + \Sigma_i w_{k,i} x_{it}, \quad (5.1)$$

where n is the net input of the kth neuron in the hidden layer and x the ith input to the respective neuron. Subscript t denotes a point in time. $W_{k,i}$ represents the weight of the kth neuron which is attached to the ith input of the neuron. Thus it can be concluded that $\Sigma_i w_{k,i} x_{it}$ is the weighted sum of all incoming inputs x_i at neuron k. This sum is modified by the bias $w_{k,0}$ of the kth neuron. Provided the threshold value is reached we write further

$$N_{k,t} = (1 + \exp(-n_{k,t} g))^{-1}, \quad (5.2)$$

which is the net input $n_{k,t}$ squashed through the transfer function of the kth neuron in the hidden layer.

$$y_t = \gamma_0 + \Sigma_k \gamma_k N_{k,t}, \quad (5.3)$$

is the output equation where γ_k is the weight matrix between the kth neuron of the hidden layer and the output layer. γ_0 is the bias of the output neuron. The summation sign indicates that the output y_t of the network depends on each neuron k of the hidden layer.

Multi-layer perceptrons analyze diversity and so they are used for applications in high dimensional spaces [Zim05, p.29].

5.2 The radial basis function

The radial basis function network uses the Gaussian density function as its non-linear transformation device [Mcn05, p.28]. This is the major differences between the two types of feedforward networks presented in this chapter. As figure 5.3 shows, the architecture of the network is exactly like the one of the multi-layer perceptron however, the activation of the neurons is performed differently.

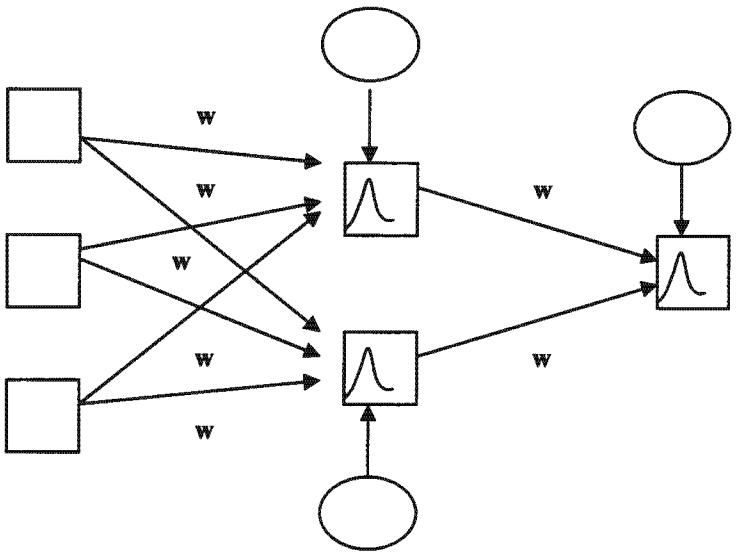


Figure 5.3 The architecture of a radial basis function

Neurons do use the weighted sum of inputs but not the sigmoid transfer function as suggested in figure 3.1. Instead, each neuron implements a Gaussian function with predefined mean μ and variance σ^2 [Mcn05, p.28]. According to the input, a probability is computed with higher values around the mean as the following graphic reveals.

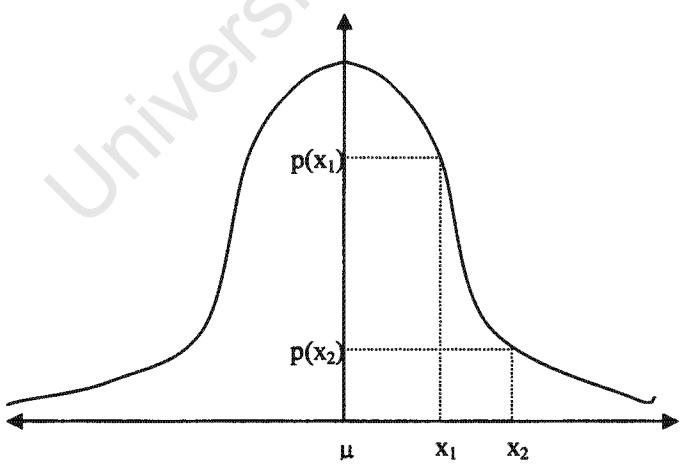


Figure 5.4 Input output transformation by the Gaussian

x_1 and x_2 are two representative input values, which generate probabilities $p(x_1)$ and $p(x_2)$ respectively. The graphic illustrates that the closer the input is to the mean, the higher the generated probability. All probabilities are finally summed up and compared to a threshold. Again, the closer an input value

comes to the mean, the more "weight" it gains in the summation of inputs [Tar98, p.28]. In other words, the contribution of an input to the activation of a neuron is determined by the distance between itself and the center of the basis function. Hence, figure 3.2 can be amended for radial basis functions to,

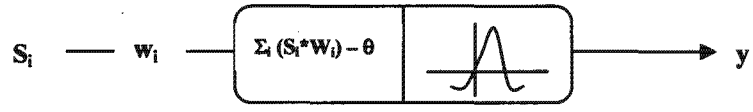


Figure 5.5 The artificial neuron for radial basis function networks

As the input moves away from the given center μ, the neuron output rapidly drops to zero [BDH05, p.19-2]. Figure 5.6 depicts the multivariate case of a Gaussian function. Note that the input points denoted by triangles contribute less to the activation than the squares.

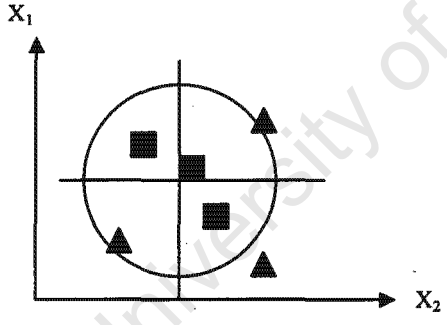


Figure 5.6 Contribution of inputs to the activation of radial basis neurons

To forecast the output, the different Gaussian transformations are combined in linear fashion [Mcn05, p.28]. We write,

$$n_t = w_0 + \Sigma_i x_{i,t} w_i , \qquad (5.4)$$

where the net input n_t is the linear transform of the inputs x_i and w_0 the bias. Now,

$$R_{k,t} = \Phi(n_t, \mu_k) = 1 / \sqrt{2\pi\sigma} \exp(-(n_t - \mu_k)/ \sigma^2). \qquad (5.5)$$

According to the number of neurons in the hidden layer, k different radial basis functions are chosen with μ_k and σ_k^2 different for each neuron k.
The output y_t of the network is a weighted sum of the transformed input, including a bias γ_0 such that

$$y_t = \gamma_0 + \Sigma_k \gamma_k R_{k,t}. \quad (5.6)$$

Radial basis functions are a particular type of feedforward networks which analyze similarities rather than diversity (compared to MLP). Hence, they are inferior to multi-layer perceptrons when it comes to high dimensional problems [Zim05, p.29]. radial basis networks have at most two hidden layers [Tar98, p.28] [BDH05, p.19-2].

5.3 Summary of important differences

	MLP	RBF
Activation	Sigmoid or tanh function	Gaussian function
Application	Superior w.r.t. high dimensional problems since analysis of diversity.	Inferior w.r.t. high dimensional problems since analysis of similarity.
Layers	Undefined number	Max. two hidden layers
Training Speed	slow	fast

Table 5.1 Differences between MLP and RBF

5.4 Decreasing model uncertainty

One problem inherent to each decision model is the level of uncertainty concerning the output. With respect to neural networks, differing results could be due to instabilities in the learning process or to the network design [NZ98, p.9]. One possible solution to this problem is an approach which is referred to as the Monte Carlo method. The principle of this method is, to average the output of several networks and so getting better and more stable results [NZ98, p.9]. Figure 5.7 depicts the architecture of an averaged network.

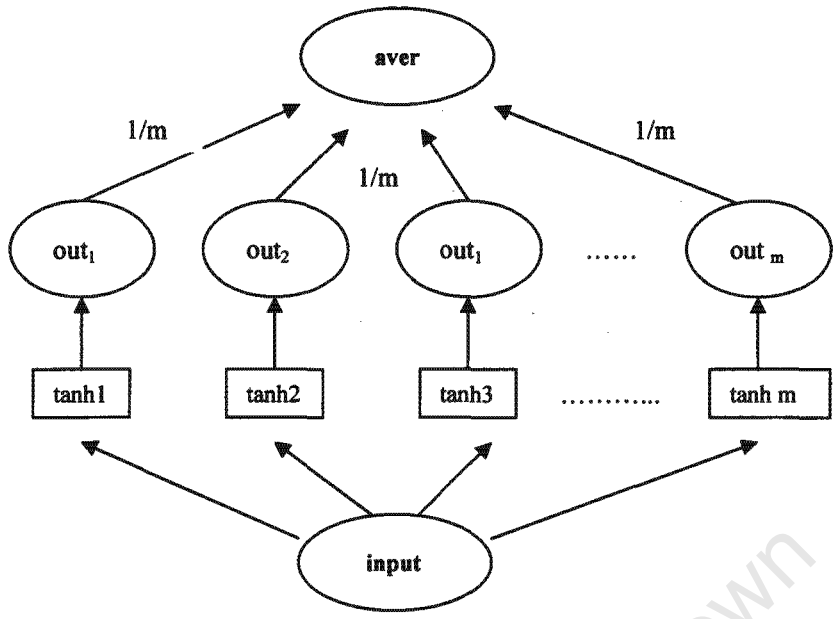


Figure 5.7 Monte Carlo network

The computation process passes two phases. In the first phase, the same input is fed into the single sub-networks represented by the squared boxes. Each sub-network reaches a different local minimum, so that different solutions are learnt of the same task [Zim05, p.34].

In the second phase the sub-solution are averaged out by the factor 1/m where m stands for the total number of solutions or sub-networks respectively [Zim05, p.34]. Subsequently the error function of an averaged network can be written as

$$E_{aver} = 1/T \, \Sigma_T \, [out_{aver} - target]^2$$

$$= 1/T \, \Sigma_T \, [(1/m \, \Sigma_i \, out_i) - target]^2$$

$$= 1/T \, \Sigma_T \, [1/m \, \Sigma_i \, out_i - target]^2 . \qquad (5.7)$$

Extracting the fraction from the brackets yields

$$E_{aver} = 1/m^2 \; 1/T \, \Sigma_T \, \Sigma_i \, (out_i - target)^2$$

$$= 1/m \; 1/m \, \Sigma_i \; 1/T \, \Sigma_T \, (out_i - target)^2, \qquad (5.8)$$

where $1/m \Sigma_i 1/T \Sigma_T (out_i - target)^2$ is the average of the sub-models and $1/T \Sigma_T (out_i - target)^2$ the error of the sub-models. Given uncorrelated errors, we come to

$$E_{aver} = 1/m \text{ average}(E_i) \quad (5.9)$$

which accomplishes the proof that the final output of the model is the average of the outputs of each sub-model.

6 Recurrent (feedbackward) neural networks

Unlike the feedforward approach, recurrent neural networks allow the neurons to depend not only on the input variables x but also on their own lagged values [Mcn05, p.34], i.e. the signals loop through the network [SS96, p.9]. Any previous model error is treated as an additional input. While learning, the model error functions as an external shock and is used to determine the model dynamics. Hence, a memory dimension is implemented in recurrent networks [NZ00, p.71] and thus time evidently plays an important role. These facts of the case are subject of chapter 6.1 which explains the integration of time, i.e. dynamics into the network architecture. In order to predict future developments, the network has to be unfolded in time. Chapters 6.2 and 6.3 depict two different kinds of unfolding which differ in their respective time horizons.

6.1 The recurrent network as a dynamic system

Following equations identify a recurrent, time unfolding system in a general sense. Equation (6.1) represents the state transition, equation (6.2) the output [Zim05, p.55] [NZ00, p.73].

$$s_t = f(s_{t-1}, u_t), \quad (6.1)$$

$$y_t = g(s_t). \quad (6.2)$$

Equation (6.1) is the mapping from the previous internal hidden state s_{t-1} to the new state s_t [NZ00, p.73]. This mapping is processed by a function f which also includes the external input u_t of the new state. Hence, the characterization of recurrent networks as a system which integrates inputs and lagged values becomes clear. Equation (6.2) describes the computation of the network output y_t performed by function g. Graphically a dynamic system can be illustrated by the following sketch,

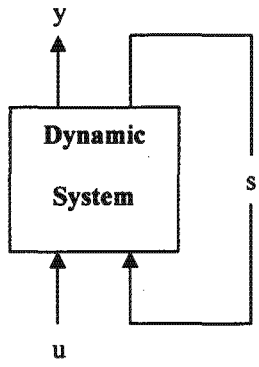


Figure 6.1 Plot of a dynamic system

where s is a lagged input.

As a next step, we define functions f and g as procedures conducted by neural networks using weight vectors v and w as additional parameters [NZ00, p.75]. We write,

$$s_t = NN(s_{t-1}, u_t, v), \tag{6.3}$$

$$y_t = NN(s_t, w), \tag{6.4}$$

such that

$$1/T \, \Sigma_T (y_t - target)^2 \rightarrow \min_{v,w}! \tag{6.5}$$

Equations (6.3) and (6.4) can be translated into neural network architecture where parameters A, B and C are respective weights. Thus we come to

$$s_t = \tanh(A\, s_{t-1} + B\, u_t), \tag{6.6}$$

$$y_t = C\, s_t. \tag{6.7}$$

Given that, figure 6.1 can be enhanced to

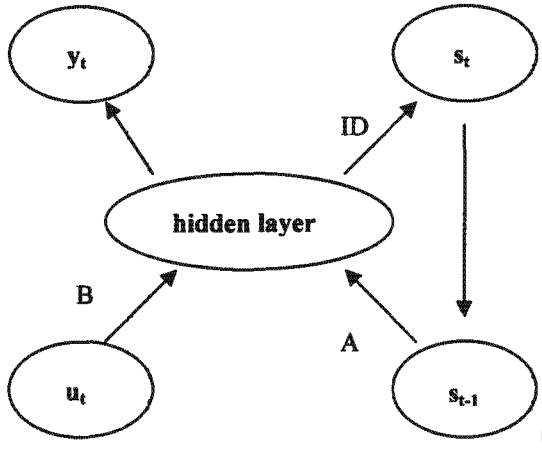


Figure 6.2 Neural network architecture of a dynamic system

where ID is an identity matrix.

6.2 Finite unfolding

Unfolding in time generates a non-local structure which opposes to the neural network paradigm of localized architecture and algorithms [Zim05, p.59]. However, localization is important in order to guarantee an unrestricted scaling of the network [Zim05, p.9]. The solution to this problem is the use of shared weights [NZ00, p.76]. Shared weights provide the same weights value at each time step. This can be implemented in one weight matrix for all neurons instead of defining matrices for each single one. Whenever a neuron of the network processes a linear transformation, it can instantly access the shared matrix.

Figure 6.3 depicts a recurrent network.

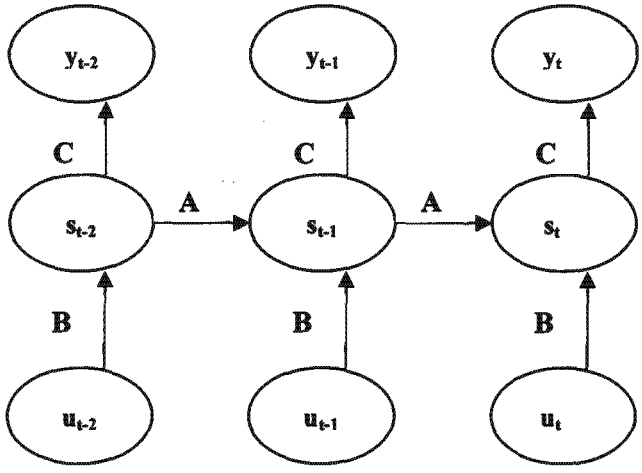


Figure 6.3 Recurrent neural networks: finite unfolding

As we can see, the network unfolds starting from t-2 and is truncated in t. Equation (6.5) computes the individual errors for y_{t-2}, y_{t-1}. By superposition, i.e. the addition of past time steps to the network, the final error in time t decreases to a minimum. Adding more superpositions the error level might start reverting up again. Thus the minimal error determines the number of time steps [NZ00, p.76]. Figure 6.4 illustrates this, where ts* is the optimal number of time steps and E the error.

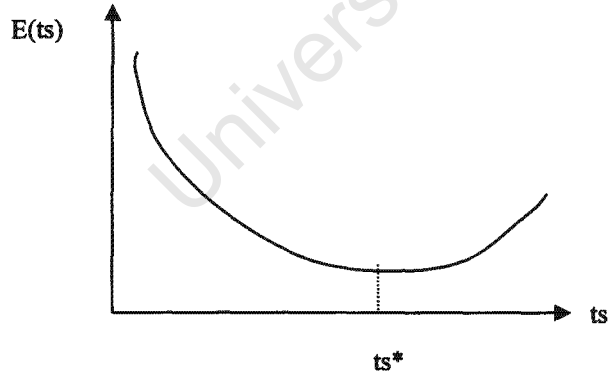


Figure 6.4 Model size and error

Eventually, the term "finite unfolding" refers to the truncation of the network in today, i.e. time t. No future predictions are performed so that this type of network is the most basic one.

6.3 Overshooting

In financial analysis however, forecasts of future developments are central tasks. Hence, a decision making system has to show the ability of working out future time steps. Overshooting is a method with which recurrent networks can be extended into the future as figure 6.5 illustrates.

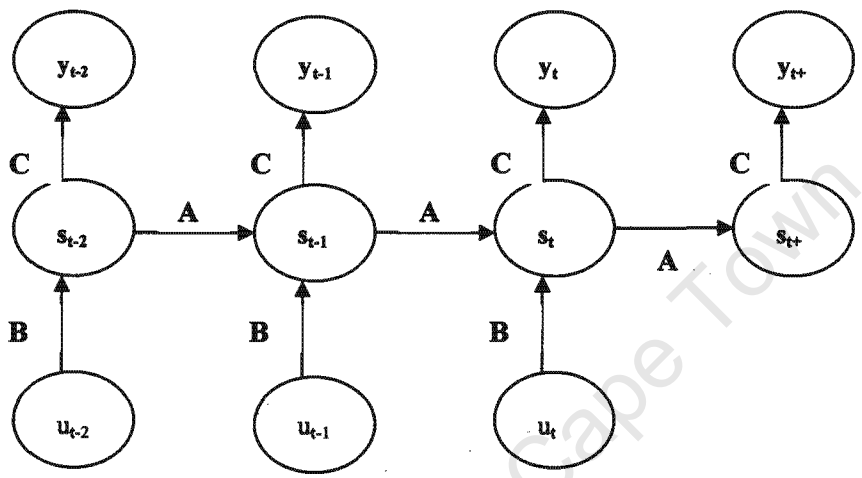


Figure 6.5 Recurrent neural networks: overshooting

Overshooting adds to the externally driven part t-2 to t an autonomous extension t+1. Since the extension describes the future, no external input (still unknown) flows into the computation. Hence its naming "autonomous" [NZ00, p.71]. Just like in the finite case, the number of time steps is determined by the minimal error.

A final point of interest shall be mentioned. Recurrent networks have to be initialized since their start values equal to zero. Hence, an artificial ignition has to be introduced into the network. The artificial character makes this input appear as noise. Noise however, is an undesired element. It causes distortions in the output by negatively affecting the error function [NZ98, p.29]. This problem is solved by the implementation of a contractive matrix which squeezes noise out of the process [Zim05, p.61]. A matrix of this kind is matrix A in figures 6.3 and 6.5.

6.4 The error correction neural network

The idea behind error correction neural networks is it to use the previous model error as an additional input into the network. This may help to overcome the problem that relevant external drivers can hardly be identified or that data are too noisy [Nyg04, p.14].

Again, equations (6.8) and (6.9) stand for state transition and output.

$$s_t = f(s_{t-1}, u_{t-1}, yt - target_t), \tag{6.8}$$

$$y_t = g(s_t) \tag{6.9}$$

respective the optimization problem

$$E = 1/T \sum_T (y_t - target_t)^2 \rightarrow \min_{f,g}! \tag{6.10}$$

Equations (6.8) to (6.10) can be transformed in neural network archtecture [Zim05, p.66].

$$s_t = \tanh(A\, s_{t-1} + B\, u_{t-1} + D \tanh(C\, s_{t-1} - target_{t-1})), \tag{6.11}$$

$$y_t = C\, s_{t-1}, \tag{6.12}$$

$$E = 1/T \sum_T (y_t - target_t)^2 \rightarrow \min_{A,B,C,D}! \tag{6.13}$$

where A, B, C and D are respective weight matrices. Equations (6.8) and (6.11) reveal the integration of the error by $(y_t - target_t)$. The following graphic depicts an error correction network where (6.11) is modified to

$$s_t = \tanh(A\, s_{t-1} + B\, u_{t-1} + D \tanh Z_t), \tag{6.14}$$

Equations (6.12) to (6.14) are represented by the gray shaded nodes.

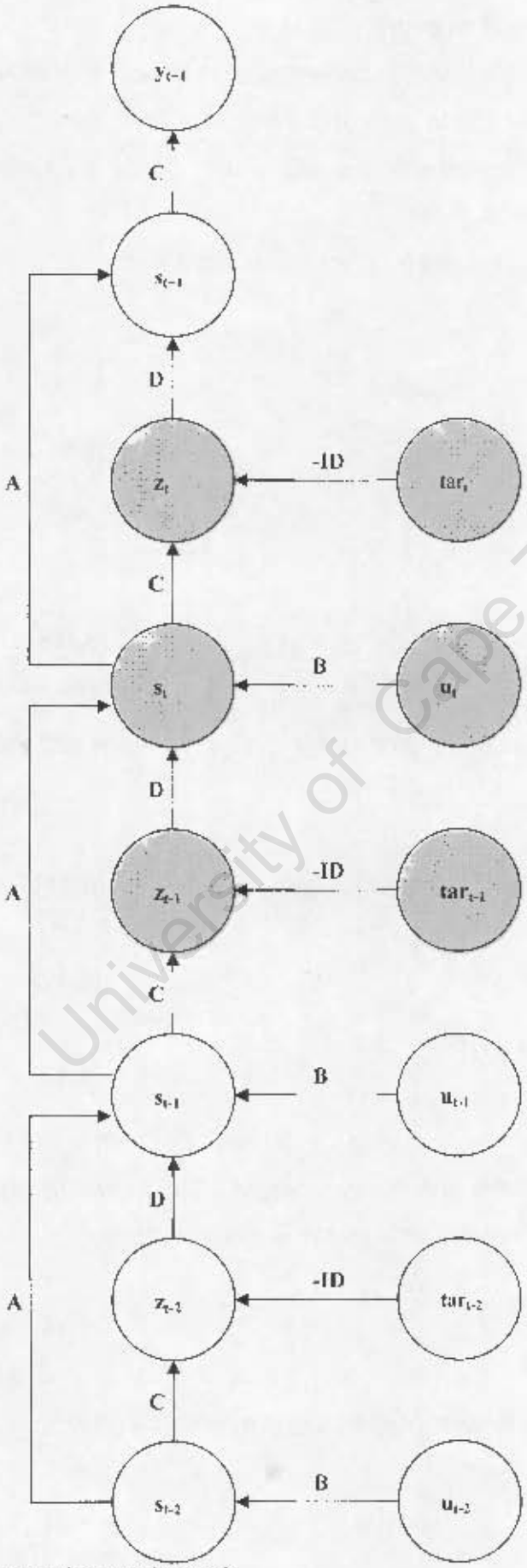


Figure 6.6 Error correction neural network

7 Practical application

The following chapter will show the results of the simulations conducted with SENN. In the first part the benchmark is introduced. Benchmarks are important features in testing since they give evidence about the performance of the computation such as efficiency and speed. In addition, benchmarks provide information about computation accuracy, potential enhancements and alternative techniques [Azo94, p.91]. In the following, tables and graphs of the computational results are displayed along with their respective interpretation.

7.1 The benchmark

The data which underlay the computation are stock data. Computation of interest rates, exchange rates, bond yields or implied volatilities are also possible but shall be excluded in this study. The company which stock is used for computation is ABSA, one of the leading banks in South Africa and an important value in the Johannesburg Stock Index.

The period of interest is the year 2005. Data are available from the 3rd of January to the 30th of December. For missing data, linear interpolation was applied, where p is the stock price,

$$p_t = (p_{t-1} + p_{t+1}) / 2 . \qquad (7.1)$$

However, simple prices are not used for the computation since nonlinear estimation requires data to be stationary. In order to obtain stationarity, plain stock prices are preprocessed in the following way,

$$\Delta p_t = \ln(p_t) - \ln(p_{t-1}). \qquad (7.2)$$

This transformed data series is free of any dynamic and so suitable for nonlinear processing.

A last important requirement is the scaling of inputs. Scaling is crucial for computation since very high or low outliers could cause computation overflows. This means that the system attributes the value zero to outliers

which can lead to false computational results. Depending on the scaling interval, different methods of scaling are possible. For elements $x \in [0,1]$ or $[-1,1]$,

$$x'_{k,t} = [x_{k,t} - \min(x_k)] / [\max(x_k) - \min(x_k)] \quad (7.3)$$

or,

$$x''_{k,t} = 2 * [x_{k,t} - \min(x_k)] / [\max(x_k) - \min(x_k)] - 1. \quad (7.4)$$

Scaling has the effect that outliers are attributed with the value 1 or 0. More on scaling can be found in McNelis [Mcn05, pp. 64].

In order to conduct the neural computation the data field has to be partitioned into a training set, a validation set and a generalization set. In the training period the neural networks learns patterns which occur in the time series. The learnt patterns are tested against data in the validation set. The validation set is a subset of the training set. All data which are not in the training set belong to the generalization set. This is a set of data on which the learning progress is tested. The less the deviation between the simulation and this data series, the better the model. The following graphic illustrates the important relation between the three introduced sets of data.

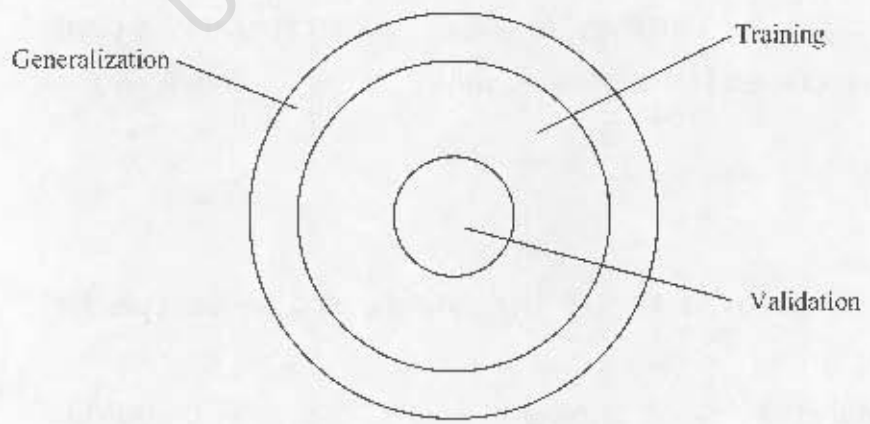


Figure 7.1 Data partition for neural computing

7 Practical application

The following chapter will show the results of the simulations conducted with SENN. In the first part the benchmark is introduced. Benchmarks are important features in testing since they give evidence about the performance of the computation such as efficiency and speed. In addition, benchmarks provide information about computation accuracy, potential enhancements and alternative techniques [Azo94, p.91]. In the following, tables and graphs of the computational results are displayed along with their respective interpretation.

7.1 The benchmark

The data which underlay the computation are stock data. Computation of interest rates, exchange rates, bond yields or implied volatilities are also possible but shall be excluded in this study. The company which stock is used for computation is ABSA, one of the leading banks in South Africa and an important value in the Johannesburg Stock Index.

The period of interest is the year 2005. Data are available from the 3rd of January to the 30th of December. For missing data, linear interpolation was applied, where p is the stock price.

$$p_t = (p_{t-1} + p_{t+1}) / 2 . \qquad (7.1)$$

However, simple prices are not used for the computation since nonlinear estimation requires data to be stationary. In order to obtain stationarity, plain stock prices are preprocessed in the following way.

$$\Delta p_t = \ln(p_t) - \ln(p_{t-1}). \qquad (7.2)$$

This transformed data series is free of any dynamic and so suitable for nonlinear processing.

A last important requirement is the scaling of inputs. Scaling is crucial for computation since very high or low outliers could cause computation overflows. This means that the system attributes the value zero to outliers

which can lead to false computational results. Depending on the scaling interval, different methods of scaling are possible. For elements x ϵ [0,1] or [-1,1],

$$x'_{k,t} = [x_{k,t} - \min(x_k)] / [\max(x_k) - \min(x_k)] \quad (7.3)$$

or,

$$x''_{k,t} = 2 * [x_{k,t} - \min(x_k)] / [\max(x_k) - \min(x_k)] - 1. \quad (7.4)$$

Scaling has the effect that outliers are attributed with the value 1 or 0. More on scaling can be found in McNelis [Mcn05, pp. 64].

In order to conduct the neural computation the data field has to be partitioned into a training set, a validation set and a generalization set. In the training period the neural networks learns patterns which occur in the time series. The learnt patterns are tested against data in the validation set. The validation set is a subset of the training set. All data which are not in the training set belong to the generalization set. This is a set of data on which the learning progress is tested. The less the deviation between the simulation and this data series, the better the model. The following graphic illustrates the important relation between the three introduced sets of data.

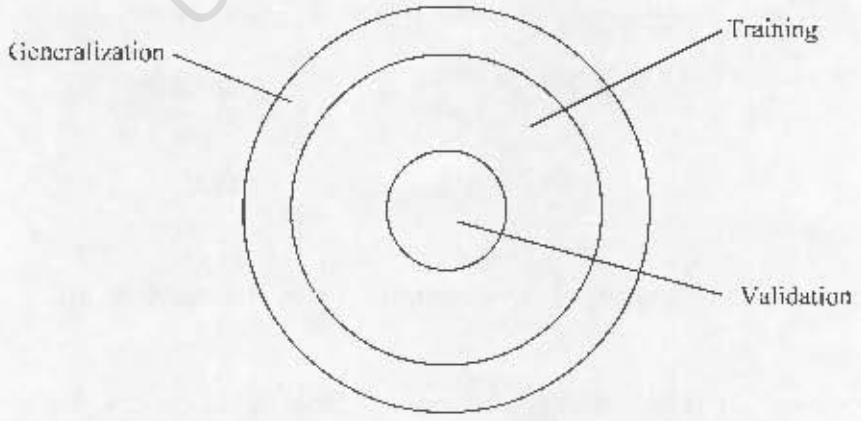


Figure 7.1 Data partition for neural computing

7.2 Computational results

The computation for the same interval of data was conducted through the use of each of the presented network architectures:

- Feedforeward network,
- Monte Carlo network,
- Recurrent network,
- Error Correction network.

In the following, results will be illustrated graphically and interpretations given. Subject of the practical study will be whether or not and how sophisticated neural networks indicate right trends in stock prices and how much the results of respective models deviate from the real data series. Furthermore it will be examined how well the networks are suitable for day trading, i.e. can they reliably predict the direction of the stock price from one day to the following.

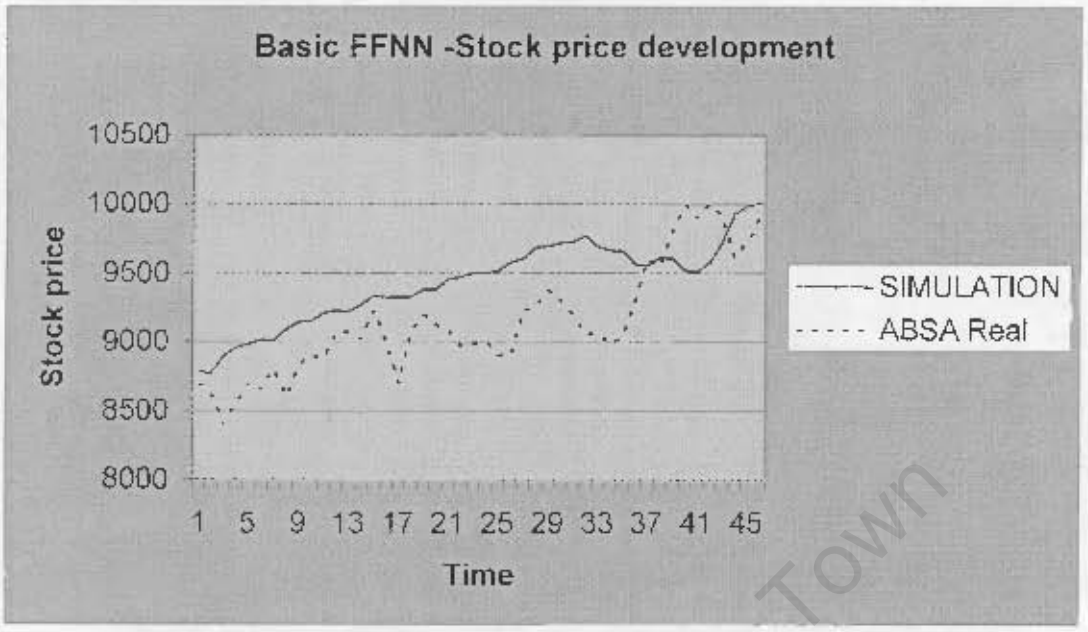
However, the monetary side shall not be neglected. Two trading strategies, a buy-and-hold approach and a neural approach will be compared. Neural approach shall stand for a strategy by which the network's forecast for the next day is compared to the real price change. Is the prediction positive, a buy or hold signal is given.

Since in real world trading involves transaction costs, a second comparison will elaborate the comparative advantage costs which are computed according to,

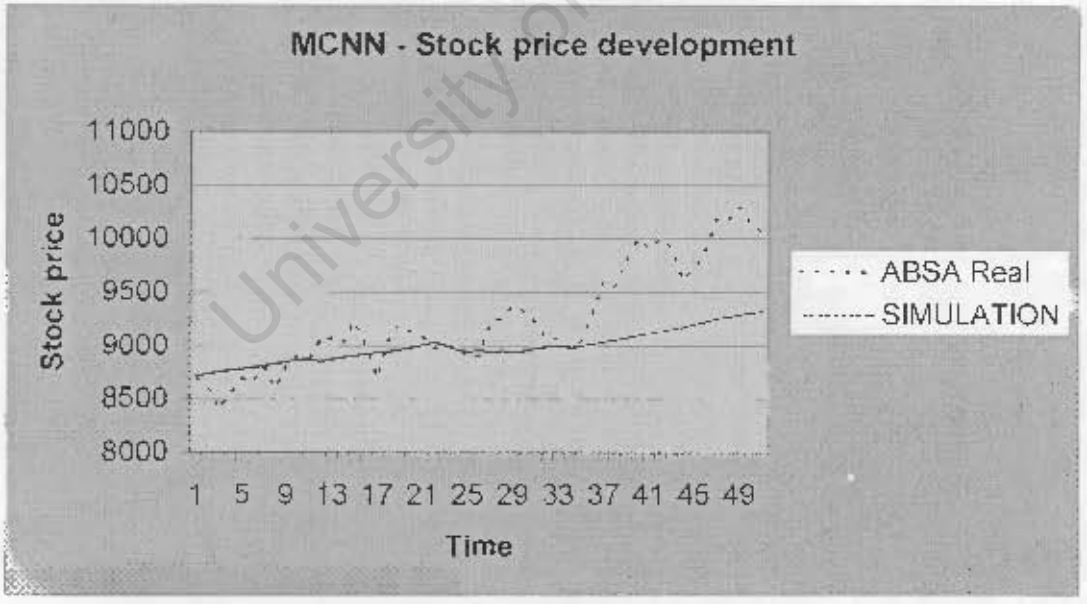
$$CAC = (NNR - BHR) / NoT, \qquad (7.5)$$

where NNR denotes the return of the neural network strategy, BHR the return of a buy-and-hold strategy and NoT the number of trades. Hence the comparative advantage costs indicate the maximal costs under which the neural approach is not yet inferior to a buy-and-hold strategy.

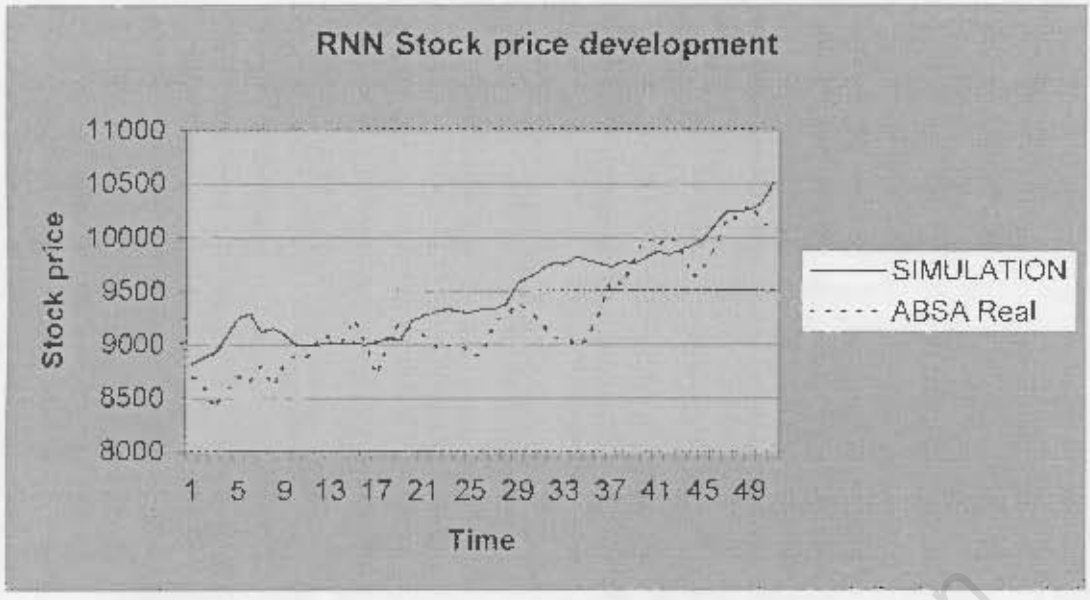
7.2.1 Trends and deviation from real data



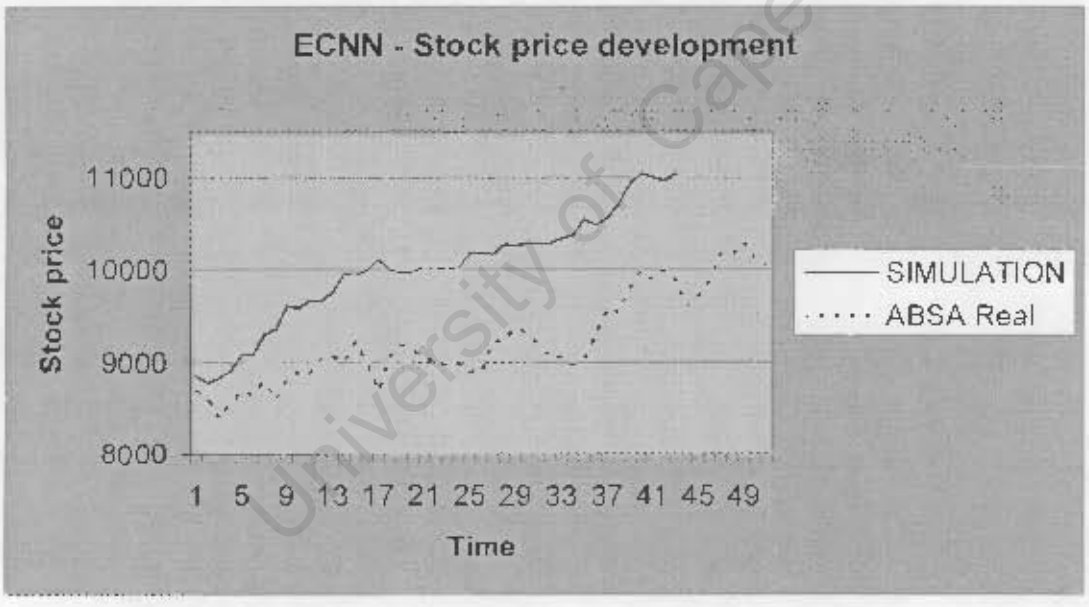
(a) real and simulated stock price development applying basic feedforeward methods



(b) real and simulated stock price development applying Monte Carlo methods



(c) real and simulated stock price development applying basic recurrent methods



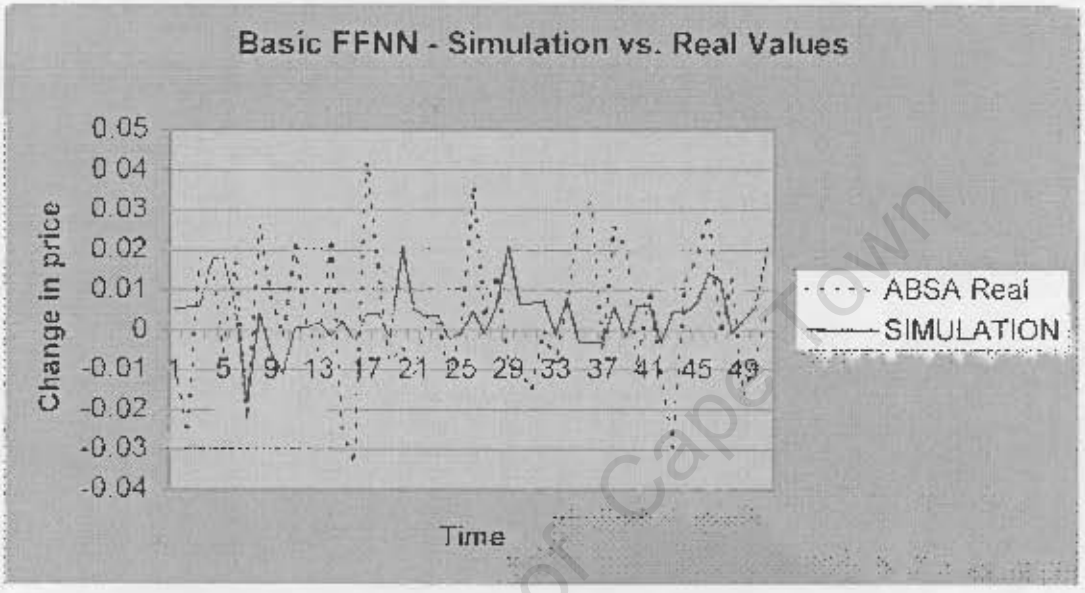
(d) real and simulated stock price development applying error correction methods

Graphic 7.1 Comparison of stock price developments for FFNN, MCNN, RNN and ECNN

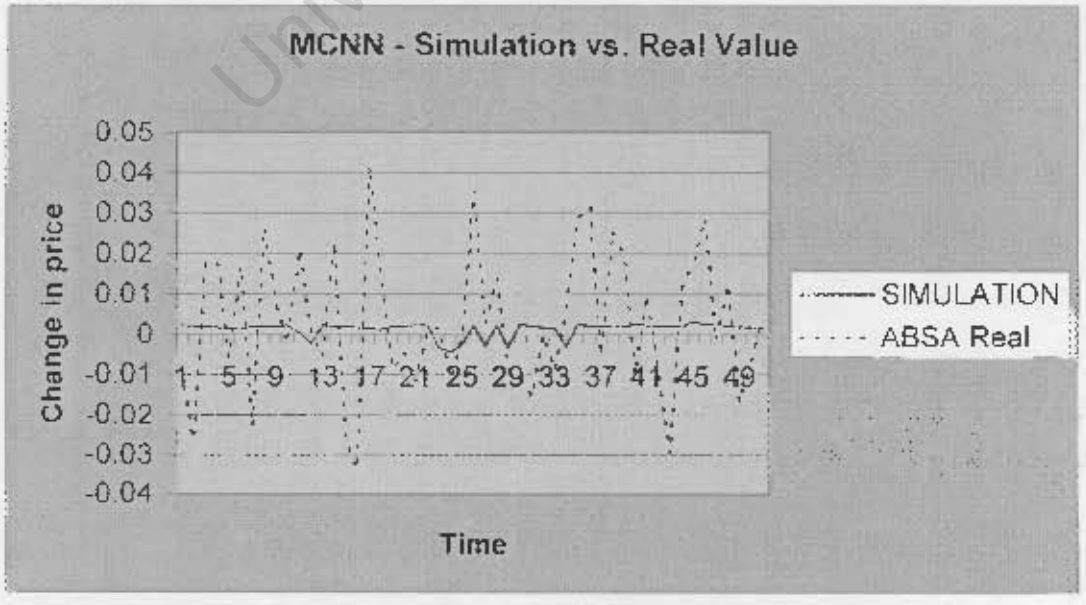
It can be seen, that all four networks are capable to work out the generally up sloping trend in the stock price development of ABSA. However, differences in the accuracy of the different predictions are obvious.

Basic feedforeward (FFNN) and error correction (ECNN) networks seem to overestimate the stock development whereas recurrent (RNN) and Monte Carlo (MCNN) networks rather indicate a smooth average. From time point

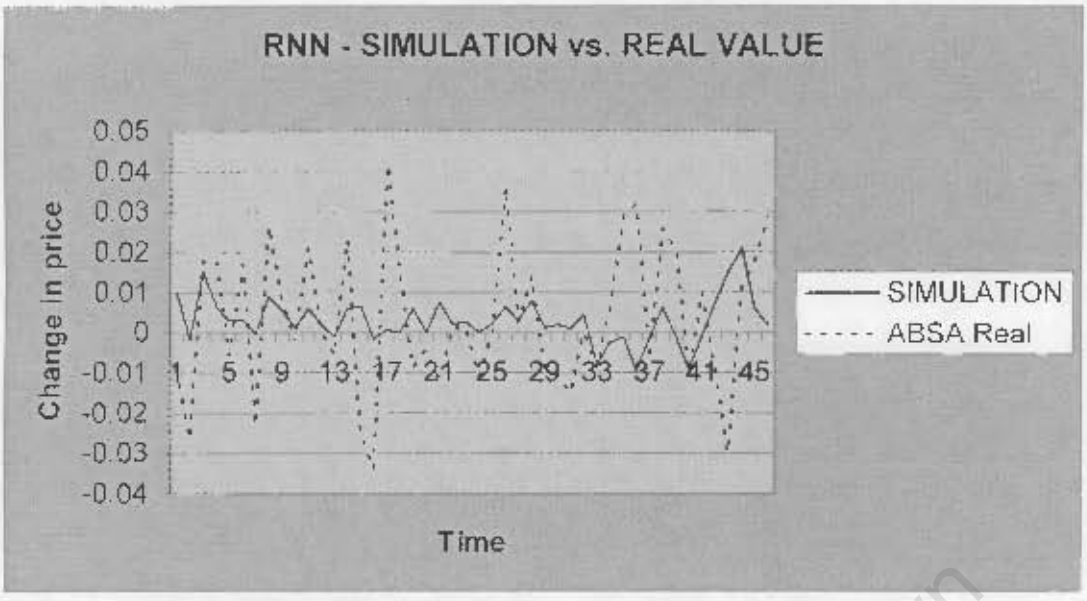
26 on however, MCCNN underestimate the stock trend. Since MCNN compute the average of a defined number of feedforward networks, it becomes clear why the plotted trend line is rather smooth. The reason for why different types of networks yield different levels of accuracy depends on the level of precision in the simulation of stock price changes. Graphic 7.2 compares simulated returns to real returns.



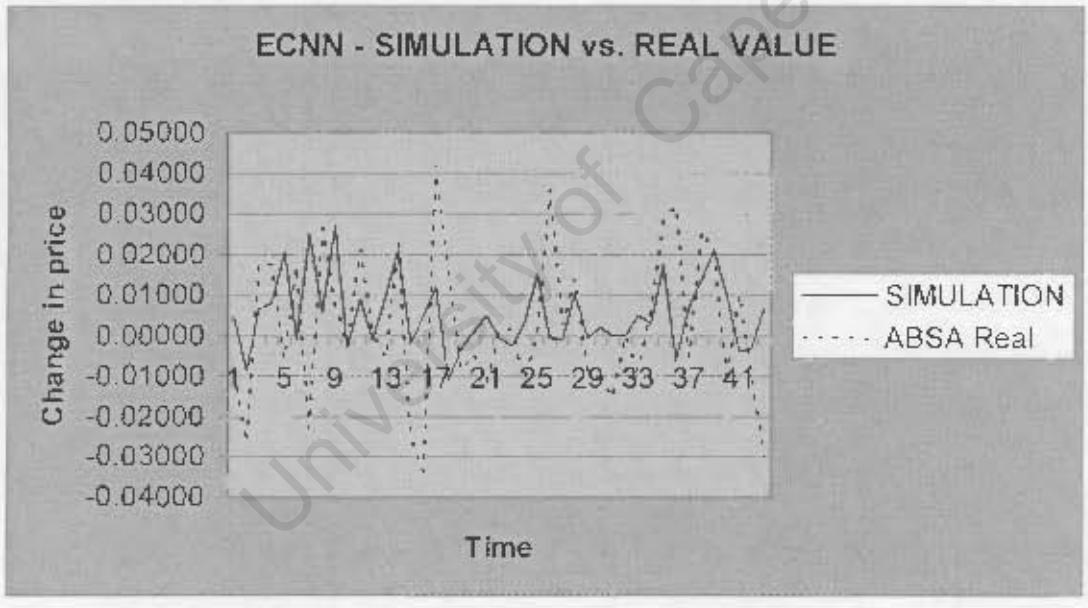
(a) real and simulated stock price changes applying basic feedforeward methods



(b) real and simulated stock price changes applying basic Monte Carlo methods



(c) real and simulated stock price changes applying basic recurrent methods



(d) real and simulated stock price changes applying error correction methods

Graphic. 7.2 Comparison of simulated and real changes in stock prices

Graphic 7.2 illustrates heavy amplitudes in the simulations of ECNN and FFNN whereas the amplitudes of the RNN and the MCNN are less accentuated. This is the key for the understanding of the degree of volatility simulated in graphic 7.1. Regarding MCNN in graphic 7.2. we see little peaks in the solid line. This is the reason for the smooth stock price simulated by MCNN in graphic 7.1.

In addition, most of the time the simulations are in the positive part of the coordinate system. This means positive changes in prices and is thus the reason for the overall positive trends indicated in graphic 7.1. ECNN as the most advanced networks applied in this study seem to imitate the stock price movements with the highest accuracy. This optical impression however, is not confirmed by the following table which illustrates the total deviation between the real values and the respective simulations.

	Total deviation between real values and simulations
FFNN	0.00042
MCNN	0.00241
RNN	0.00036
ECNN	0.00045

Table 7.1 Total deviation of returns

As expected, the total deviation is highest for the MCNN simulation. RNN on the other hand show the least deviation from the real time series. It shall also be mentioned that the deviation of the basic feedforward networks is unexpectedly low. The question is now whether or not the deviation from the real values is a sensible indicator for the choice of a certain model. To test this hypothesis the next chapter implements two trading scenarios. As results of these scenarios we will be able to determine which model is the best to use and whether or not neural networks contribute to the improvement of trading.

7.2.2 Network test under trading conditions

The following tables illustrate a 20-day trading period. A plus in the column "correlation" indicates that the network guessed the right direction of the stock price. The shaded areas indicate sell signals. Since the learning success of neural networks shrinks with time a 10- and 20-day period will be examined.

Date	Δ sim. price	Δ real price	real price	correlation
day 1	0.005	-0.0086	8700	
day 2	0.0055	-0.0264	8625	+
day 3	0.0056	0.0177	8400	-
day 4	0.0175	0.0174	8550	-
day 5	0.018	-0.0057	8700	+
day 6	0.005	0.0171	8650	+
day 7	-0.0185	-0.0228	8799	+
day 8	0.004	0.0264	8600	+
day 9	-0.006	0.0078	8830	+
day 10	-0.011	-0.0022	8900	-
day 11	0.0001	0.0212	8880	-
day 12	0.0005	0	9070	-
day 13	0.0015	-0.0055	9070	+
day 14	-0.0016	0.023	9020	+
day 15	0.002	-0.0252	9230	-
day 16	-0.003	-0.0339	9000	-
day 17	0.0038	0.0416	8700	+
day 18	0.004	0.0142	9070	-
day 19	-0.0035	-0.0087	9200	+
day 20	0.0205	-0.0044	9120	-

Table 7.2 Trading with FFNN

Regarding the 10-day period, 67% of the time the network guesses the right direction of the stock movement compared to only 40% percent in the second period. This finding coincides with general expectations that learning deteriorates over time.

Compared to a buy-and-hold strategy, the neural approach yields better results for both a 10-day and a 20-day period. For 10-days the buy-and-hold strategy promises 2.07% or 4.08% while the FFNN yields 3.79% for 10 days and 7.98% for 20 days. Including fixed transaction costs for trading the neural network approach is not worse than a buy-and-hold strategy if the costs per trade do not exceed 0.573% of the profits for a 10-day period and 0.39% of the profits for a 20-day period. Hence, these rates are called comparative advantage rates.

Date	Δ sim. price	real price	real price	correlation
day 1	0.00221	-0.0086	8700	
day 2	0.00203	-0.0264	8625	+
day 3	0.00187	0.0177	8400	+
day 4	0.00170	0.0174	8550	+
day 5	0.00153	-0.0057	8700	+
day 6	0.00136	0.0171	8650	-
day 7	0.00203	-0.0228	8799	-
day 8	0.00210	0.0264	8600	+
day 9	0.00190	0.0078	8830	+
day 10	0.00221	-0.0022	8900	-
day 11	0.00053	0.0212	8880	-
day 12	-0.00227	0	9070	+
day 13	0.00223	-0.0055	9070	-
day 14	0.00202	0.023	9020	-
day 15	0.00187	-0.0252	9230	-
day 16	0.00169	-0.0339	9000	+
day 17	0.00154	0.0416	8700	-
day 18	0.00136	0.0142	9070	+
day 19	0.00203	-0.0087	9200	-
day 20	0.00210	-0.0044	9120	+

Table 7.3 Trading with MCNN

In the 10-day period, 67% of the daily trends are outguessed by the network but only 44% for the second period. This considerably worse result of MCNN in comparison to basic FFNN can be explained by the effect of averaging out different solutions. Since the network gives only one sell signal (day 12) where the return is zero, there is no difference between a buy-and-hold approach and the use of the network. Both strategies yield 2.07% for the 10-day period and 4.3% for the 20-day period. Logically, taking transaction costs into consideration turns the MCNN into an inferior strategy. Hence, the comparative advantage rate is zero.

Date	Δ sim. price	real price	real price	correlation
day 1	0.01	-0.0086	8700	
day 2	-0.0018	-0.0264	8625	+
day 3	0.015	0.0177	8400	-
day 4	0.0065	0.0174	8550	+
day 5	0.003	-0.0057	8700	+
day 6	0.0032	0.0171	8650	+
day 7	-0.0005	-0.0228	8799	+
day 8	0.009	0.0264	8600	+
day 9	0.0057	0.0078	8830	-
day 10	0.001	-0.0022	8900	+
day 11	0.006	0.0212	8880	+
day 12	0.002	0	9070	+
day 13	-0.001	-0.0055	9070	+
day 14	0.006	0.023	9020	-
day 15	0.0056	-0.0252	9230	-
day 16	-0.0015	-0.0339	9000	+
day 17	0.0007	0.0416	8700	+
day 18	0	0.0142	9070	+
day 19	0.006	-0.0087	9200	-
day 20	0	-0.0044	9120	-

Table 7.4 Trading with RNN

Recurrent networks guess the right direction of stock movements in 77% of the cases for the first 10 days. This performance deteriorates slightly during the second half of the time interval. However, with 68% success probability recurrent networks still provide reliable indications of the future trends.

Regarding trading, RNN outperform by far a buy-and-hold approach. For the first 10 days of the period, RNN yield 6.43% profit while the buy-and-hold approach only provides 2.07%. For the whole 20 days, RNN give 12.6% while the buy-and-hold strategy only generates 4.3%. Including trading costs it can be noted that in a 10-day interval RNN are not inferior unless the expenses per trade are less or equal to 1.09% and 1.04% for 20 days.

	Δ sim. price	real price	real price	correlation
	0.00450	-0.0086	8700	
day 2	-0.00850	-0.0264	8625	+
day 3	0.00650	0.0177	8400	+
day 4	0.00800	0.0174	8550	-
day 5	0.02050	-0.0057	8700	-
day 6	-0.00100	0.0171	8650	-
day 7	0.02500	-0.0228	8799	-
day 8	0.00600	0.0264	8600	-
day 9	0.02700	0.0078	8830	-
day 10	-0.00300	-0.0022	8900	+
day 11	0.00900	0.0212	8880	+
day 12	-0.00100	0	9070	+
day 13	0.01000	-0.0055	9070	-
day 14	0.02100	0.023	9020	+
day 15	-0.00200	-0.0252	9230	+
day 16	0.00500	-0.0339	9000	-
day 17	0.01200	0.0416	8700	+
day 18	-0.01100	0.0142	9070	+
day 19	-0.00300	-0.0087	9200	+
day 20	0.00100	-0.0044	9120	-

Table 7.5 Trading with ECNN

Although error correction networks are a sophisticated type of recurrent networks, they do not outperform basic RNN with respect to direction guessing. For the first 10-day period, only 33% of the stock moves were guessed in the right way. 77% however, for the second period. This result is intriguing but rather a coincidence than the norm.

ECNN improve trading results since they yield 3.22% for a 10-day period and 7.42% for a 20-day period. In comparison, the buy-and-hold strategy only yields 2.07% and 4.3% respectively. Considering trading costs, each transaction must not be more expensive than 0.29% (10-day) or 0.26% (20-day) of the profit.

We can conclude that neural network based strategies outperform naive buy-and-hold strategies under certain circumstances involving transaction costs. However, since transaction costs become less and less expensive due to strong competition among stock exchanges and investment brokers and due to electronic trading and information systems, these kind of costs only negatively affect profits if trades are committed in a considerably high frequency. Going away from a naive neural network strategy where each

negative forecast results in a sell signal towards a more intelligent approach can help to curb transaction costs. Such an improvement in trading could be the installation of a downward threshold. This threshold can be oriented on the transaction costs themselves. For example if a forecast is less negative than prevailing transaction costs, a hold signal is rather given.

At last it must be mentioned that the forecasting power of neural networks does not include unexpected events. Neural networks predict future developments by evaluating past patterns. Unexpected events such as the Asian crisis and the bankruptcy of Russia in the late 1990s or September 11th have a destructive impact on a pattern. They are not occurring regularly hence, are unlikely to be predicted. This however, cannot be regarded as an exclusive shortfall of neural forecasting methods. Other forecasting techniques are equally unlikely to incorporate the impact of such single and random events into their predictions.
However, it is important to be aware of the limits and constraints of a certain technique in order to interpret its computations and to assess its applicability.

8 Conclusion

The principle purpose of this research paper is the verification of the hypothesis that neural networks can be applied to positively affect the outcomes of trading. Evidence was given that neural networks are indeed applicable to markets. The reason for this is that markets, unlike the efficient market theory suggests do in fact show patterns which can be exploited. The origin of these patterns is the often irrational behaviour of market participants which are influenced and biased by their social environment. Single market agent behaviour can hardly be outguessed. Groups of individuals however, cause patterns of behaviour which are displayed by the market. Since neural networks are tools for pattern recognition they seem to be a suitable method to predict markets.
The practical part has shown that neural networks indeed outperform a buy-and-hold strategy. Yet further studies could be conducted comparing neural networks to different kinds of strategies or forecasting methods. Yao, Tan and Poh for example compared neural networks to linear forecasting

techniques and came to the result that the network approach is superior [PTY99].

It was also shown that the scope for transaction costs is rather large and that generally low fees do not affect the overall result. Intelligent trading systems can be installed to minimize trading costs with respect to the frequency of trades. In addition, each type of network was seen to be capable of predicting the general trend of a stock over a period of one and a half months. For day trading we found that RNN performed best while MCNN were least suitable. This finding is also backed up by the trading performance with respect to trading profits. While RNN outperformed other networks, MCNN yielded the smallest profits. As table 7.1 suggests these findings can be referred to the degree of deviation between the real data points and the simulated series. However, it was surprising to notice that basic recurrent neural networks outperformed the more advanced type of an error correction network. This fact could be referred to coincidence or to model building biases.

After all it takes experience and time to fully comprehend the whole functionality of neural networks and to exploit their total potential. More over, large and extended neural networks as the most powerful of their kind have not been tested in this study. Nevertheless, this research has proven that users with relatively little experience can use even simple network models to enhance their trading results.

APPENDIX A

FEEDFORWARD MODEL – TOPOLOGY FILE

```
net
   {
      //defining number of hidden layers
      const nr_hidden = 15;

      //defining layers
      cluster (                    IN) input;
      cluster (DIM(nr_hidden), HID) hidden;
      cluster (                   OUT) output;

      //defining layer connectivity
      connect (input -> hidden);
      connect (hidden -> output);

      //defining bias connectivity
      connect (bias -> hidden);
      connect (bias -> output);
   } mlp;
```

FEEDFORWARD MODEL – SPECIFICATION FILE

```
APPLICATION ABSAForecast

//defining time steps
MODE DAY WEEK 5

//defining data range
FROM 03.01.2005 TO MAX

//defining range of training set
TRAINING FROM 03.01.2005 TO 18.10.2005
//defining range of validation set
VALIDATION FROM 24.08.2005 TO 18.10.2005

//----------------------------------------------------------------------

//defining input variables
INPUT CLUSTER mlp.input
//naming of function
BEGIN absadata
//data file specification
ABSAPRICE = FILE DATA/ABSA_DATA.txt COLUMN 1
//scaling of input
INPUT = scale ((ABSAPRICE – ABSAPRICE(-1)) / ABSAPRICE(-1)
END

//----------------------------------------------------------------------

//defining target variables
BEGIN ABSAPRICE "ABSAStock Y(t->1)"
ABSAPRICE = FILE DATA/ABSA_DATA.txt COLUMN 1
TARGET = 10 * ln(ABSAPRICE(1) / ABSAPRICE)
END
```

MONTE CARLO MODEL – TOPOLOGY FILE

```
net
  {
      const_nr hidden = 15;
      const sparse    = 15;

      cluster (                    IN) input;
      //outlier handling
      cluster (DIM (LIKE input), HID) outlier;
      cluster (DIM (LIKE input), HID) square;

      //definition of hidden clusters
      cluster (DIM (nr_hidden), HID) hidden1;
      cluster (DIM (nr_hidden), HID) hidden2;
      cluster (DIM (nr_hidden), HID) hidden3;
      cluster (DIM (nr_hidden), HID) hidden4;
      cluster (DIM (nr_hidden), HID) hidden5;

      //definition of output neurons
      cluster (      OUT) output1;
      cluster (      OUT) output2;
      cluster (      OUT) output3;
      cluster (      OUT) output4;
      cluster (      OUT) output5;
      cluster (      OUT) output_average;

      //defining connections
      connect (input -> outlier, ONETOON, SETWEIGHTS(0.5));
      connect (outlier -> square, ONETOON, SETWEIGHTS(1.0));
```

```
connect (outlier -> hidden1, RANDOM(sparse));
connect (outlier -> hidden2, RANDOM(sparse));
connect (outlier -> hidden3, RANDOM(sparse));
connect (outlier -> hidden4, RANDOM(sparse));
connect (outlier -> hidden5, RANDOM(sparse));

connect (square -> hidden1, RANDOM(sparse));
connect (square -> hidden2, RANDOM(sparse));
connect (square -> hidden3, RANDOM(sparse));
connect (square -> hidden4, RANDOM(sparse));
connect (square -> hidden5, RANDOM(sparse));


connect (bias -> hidden1);
connect (bias -> hidden2);
connect (bias -> hidden3);
connect (bias -> hidden4);
connect (bias -> hidden5);

connect (hidden1 -> output1);
connect (hidden2 -> output2);
connect (hidden3 -> output3);
connect (hidden4 -> output4);
connect (hidden5 -> output5);

connect (bias -> output1);
connect (bias -> output 2);
connect (bias -> output 3);
connect (bias -> output 4);
connect (bias -> output 5);
```

```
        connect (output1 -> output_aver, DIAGONAL(0.04));
        connect (output2 -> output_aver, DIAGONAL(0.04));
        connect (output3 -> output_aver, DIAGONAL(0.04));
        connect (output4 -> output_aver, DIAGONAL(0.04));
        connect (output5 -> output_aver, DIAGONAL(0.04));
    } mlp;

setup
        {
                //attribution of activation functions to layers
                init mlp.outlier with {ActFunction {sel tanh} };
                init mlp.square with {ActFunction {sel square} };

                init mlp.hidden1 with {ActFunction {sel tanh} };
                init mlp.hidden2 with {ActFunction {sel tanh} };
                init mlp.hidden3 with {ActFunction {sel tanh} };
                init mlp.hidden4 with {ActFunction {sel tanh} };
                init mlp.hidden5 with {ActFunction {sel tanh} };
        }
```

MONTE CARLO METHOD – SPECIFICATION FILE

```
APPLICATION ABSAForecast

MODE DAY WEEK 5
FROM 03.01.2005 TO MAX
TRAINING FROM 03.01.2005 TO 18.10.2005

INPUT CLUSTER mlp.input

BEGIN absadata
        //including data source
        ABSAPRICE = FILE DATA/ABSA_DATA.txt COLUMN 1
        //data scaling
        INPUT = scale ((ABSAPRICE – ABSAPRICE(-1))/ABSAPRICE(-1))
END

TARGET CLUSTER mlp.output1
        ABSAPRICE = FILE DATA/ABSA_DATA.txt COLUMN 1
        //generating target values
        TARGET = 10 * ln(ABSAPRICE(1)/ABSAPRICE)
END

TARGET CLUSTER mlp.output2
        ABSAPRICE = FILE DATA/ABSA_DATA.txt COLUMN 1
        //generating target values
        TARGET = 10 * ln(ABSAPRICE(1)/ABSAPRICE)
END

TARGET CLUSTER mlp.output3
        ABSAPRICE = FILE DATA/ABSA_DATA.txt COLUMN 1
        //generating target values
        TARGET = 10 * ln(ABSAPRICE(1)/ABSAPRICE)
END
```

```
TARGET CLUSTER mlp.output4
        ABSAPRICE = FILE DATA/ABSA_DATA.txt COLUMN 1
        //generating target values
        TARGET = 10 * ln(ABSAPRICE(1)/ABSAPRICE)
END

TARGET CLUSTER mlp.output5
        ABSAPRICE = FILE DATA/ABSA_DATA.txt COLUMN 1
        //generating target values
        TARGET = 10 * ln(ABSAPRICE(1)/ABSAPRICE)
END

TARGET CLUSTER mlp.output_aver
        ABSAPRICE = FILE DATA/ABSA_DATA.txt COLUMN 1
        //generating target values
        TARGET = 10 * ln(ABSAPRICE(1)/ABSAPRICE)
END
```

RECURRENT METHOD – TOPOLOGY FILE

```
net
    {
            const nr_state = 15;

            cluster INPUT    (EQUIVALENT,    IN);
            cluster STATE    (EQUIVALENT, DIM(nr_state),  HID);
            cluster OUTPUT (EQUIVALENT,    OUT);

            connect STATE_STATE (STATE -> STATE);
            connect INPUT_STATE  (INPUT -> STATE);
            connect STATE_OUTPUT (STATE -> OUTPUT);
            connect BIAS_STATE (bias -> STATE);

            INPUT inputM20;
            INPUT inputM19;
            INPUT inputM18;
                .
                .
            INPUT inputM0;

            //defining start noise of RNN
            cluster (IN) startNoise;
```

```
STATE stateM19;
STATE stateM18;
        .
        .
STATE stateM0;
STATE stateP1;
        .
        .
STATE stateP10;

OUTPUT outputM19;
OUTPUT outputM18;
        .
        .
OUTPUT outputM0;
OUTPUT outputP1;
        .
        .
OUTPUT outputP10;

//defining connections of clusters
connect (inputM19 -> stateM18, INPUT_STATE);
                .
                .
connect (inputM0 -> stateP1, INPUT_STATE);

connect (startNoise -> stateM19, ONETOONE,
        STARTWEIGHTS(1.0));

connect (stateM19 -> stateM18, STATE_STATE);
                .
                .
connect (stateP9 -> stateP10, STATE_STATE);
```

```
            connect (stateM19 -> outputM19, STATE_OUTPUT);
                                    .
                                    .
            connect (stateP10 -> outputP10, STATE_OUTPUT);

            connect (bias -> stateM19, BIAS_STATE);
                                    .
                                    .
            connect (bias -> stateP10, INPUT_STATE);
    } mlp;

setup
    {
            init rnn.stateM19 with { ACTFunction {sel tanh} };
                                    .
                                    .
            init rnn.stateP10 with { ACTFunction {sel tanh} };
    }
```

RECURRENT METHOD - SPECIFICATION FILE

```
APPLICATION ABSAForecast

//defining time steps
MODE DAY WEEK 5

//defining data range
FROM 03.01.2005 TO MAX

//defining range of training set
TRAINING FROM 03.01.2005 TO 18.10.2005
//defining range of validation set
VALIDATION FROM 24.08.2005 TO 18.10.2005

INPUT CLUSTER rnn.startNoise
BEGIN
INPUT = 0      (15 times!)
END

INPUT CLUSTER rnn.inputM19
BEGIN absadata
ABSAPRICE = FILE DATA/ABSA_DATA.txt Column 1
INPUT = scale((ln(ABSAPRICE) – ln(ABSAPRICE(-1))) Lag -19
END


INPUT CLUSTER rnn.inputM0
BEGIN absadata
ABSAPRICE = FILE DATA/ABSA_DATA.txt Column 1
INPUT = scale((ln(ABSAPRICE) – ln(ABSAPRICE(-1))) Lag -0
END
```

```
TARGET CLUSTER rnn.outputM19
BEGIN absaXM19
ABSAPRICE = FILE DATA/ABSA_DATA.txt Column 1
TARGET = 50 * ln(ABSAPRICE / ABSAPRICE(-1)) Lag -19
END

.
.

TARGET CLUSTER rnn.outputP10
BEGIN absaXP10
ABSAPRICE = FILE DATA/ABSA_DATA.txt Column 1
TARGET = 50 * ln(ABSAPRICE / ABSAPRICE(-1)) Lag 10 ASSIGN TO
        Output_channelP10
END

SIGNAL
BEGIN absaX
ABSAPRICE = FILE DATA/ABSA_DATA.txt COLUMN 1
outP1 = OUTPUT output_channelP1

.
.

outP10 = OUTPUT output_channelP10

SIGNAL = 0

SIGNAL = ABSAPRICE * exp((outP1)/50)
SIGNAL = ABSAPRICE(1)
SIGNAL = ABSAPRICE * exp((outP1 + outP2)/50)
SIGNAL = ABSAPRICE(2)

.
.
.

SIGNAL = ABSAPRICE * exp((outP1 + outP2 +....+ outP10)/50
SIGNAL = ABSAPRICE(10)
END
```

ERROR CORRECTION METHOD – TOPOLOGY FILE

```
net
    {
        const nr_state = 15;

        cluster EXTERN (EQUIVALENT, IN);
        cluster INPUT (EQUIVALENT, IN);
        cluster STATE (EQUIVALENT, DIM(nr_state), HID);
        cluster OUTPUT (EQUIVALENT, OUT);

        connect EXTERN_STATE (EXTERN -> STATE);
        connect STATE_STATE (STATE -> STATE);
        connect STATE_OUTPUT (STATE -> OUTPUT);
        connect INPUT_OUTPUT (INPUT -> OUTPUT, DIAGONAL(-1.0));
        connect OUTPUT_STATE (OUTPUT -> STATE);
        connect BIAS_STATE (BIAS -> STATE);

        EXTERN externM20;
        EXTERN externM19;
                .
                .
        EXTERN externM0;

        INPUT inputM19;
        INPUT inputM18;
                .
                .
        INPUT inputM0;

        cluster (IN) startNoise;
```

```
STATE stateM19;
STATE stateM18;
        .
        .
STATE stateM0;
STATE stateP1;
        .
        .
STATE stateP10;

OUTPUT outputM19;
OUTPUT outputM18;
        .
        .
OUTPUT outputM0;
OUTPUT outputP1;
        .
        .
OUTPUT outputP10;

connect (externM20 -> stateM19, EXTERN_STATE);
                .
                .
connect (externM0 -> stateP1, EXTERN_STATE);

connect (startNoise -> stateM19, ONETOONE, STARTWEIGHT(1.0));

connect (stateM19 -> stateM18, STATE_STATE);
                .
                .
connect (stateP9 -> stateP10, STATE_STATE);
```

```
        connect (stateM19 -> outputM19, STATE_OUTPUT);
                                .
                                .
        connect (stateP10 -> outputP10, STATE_OUTPUT);

        connect (inputM19 -> outputM19, INPUT_OUTPUT);
                                .
                                .
        connect (inputM0 -> outputM0, INPUT_OUTPUT);

        connect (outputM19 -> stateM18, OUTPUT_STATE);
                                .
                                .
        connect (outputM0 -> stateP1, OUTPUT_STATE);

        connect (bias -> stateM19, BIAS_STATE);
                                .
                                .
        connect (bias -> stateP10, BIAS_STATE);
    } ecnn;

setup
        {
           init ecnn.stateM19 with { ACTFunction {sel tanh} };
                                .
                                .
           init ecnn.stateP10 with { ACTFunction {sel tanh} };

           init ecnn.outputM19 with { ACTFunction {sel tanh} };
                                .
                                .
           init ecnn.outputM0 with { ACTFunction {sel tanh} };
        }
```

ERROR CORRECTION METHOD – SPECIFICATION FILE

```
APPLICATION ABSAForecast

//defining time steps
MODE DAY WEEK 5

//defining data range
FROM 03.01.2005 TO MAX

//defining range of training set
TRAINING FROM 03.01.2005 TO 18.10.2005
//defining range of validation set
VALIDATION FROM 24.08.2005 TO 18.10.2005

INPUT CLUSTER ecnn.startNoise
BEGIN
INPUT = 0      (15 times!)
END

INPUT CLUSTER ecnn.externM20
BEGIN absadata
ABSAPRICE = FILE DATA/ABSA_DATA.txt Column 1
INPUT = scale((ln(ABSAPRICE) – ln(ABSAPRICE(-1))) Lag -20
END

.
.

INPUT CLUSTER ecnn.externM0
BEGIN absadata
ABSAPRICE = FILE DATA/ABSA_DATA.txt Column 1
INPUT = scale((ln(ABSAPRICE) – ln(ABSAPRICE(-1))) Lag -0
END
```

```
INPUT CLUSTER ecnn.inputM19
BEGIN absadata
ABSAPRICE = FILE DATA/ABSA_DATA.txt Column 1
INPUT = 50 *ln(ABSAPRICE / ABSAPRICE(-1)) Lag -19
END

.
.

INPUT CLUSTER ecnn.inputM0
BEGIN absadata
ABSAPRICE = FILE DATA/ABSA_DATA.txt Column 1
INPUT = 50 * ln(ABSAPRICE / ABSAPRICE(-1)) Lag -0
END

TARGET CLUSTER ecnn.outputM19
BEGIN ZERO
TARGET = 0
END

.
.

TARGET CLUSTER ecnn.outputM0
BEGIN ZERO
TARGET = 0
END
```

```
TARGET CLUSTER ecnn.outputP1
BEGIN absadata
ABSAPRICE = FILE DATA/ABSA_DATA.txt Column 1
INPUT = 50 * ln(ABSAPRICE / ABSAPRICE(-1)) Lag 1 ASSIGN TO
        output_channelP1
END

.
.

TARGET CLUSTER ecnn.outputP10
BEGIN absadata
ABSAPRICE = FILE DATA/ABSA_DATA.txt Column 1
INPUT = 50 * ln(ABSAPRICE / ABSAPRICE(-1)) Lag 10 ASSIGN TO
        output_channelP10
END

SIGNAL
BEGIN absaX
ABSAPRICE = FILE DATA/ABSA_DATA.txt COLUMN 1
outP1 = OUTPUT output_channelP1

.
.

outP10 = OUTPUT output_channelP10

SIGNAL = 0

SIGNAL = ABSAPRICE * exp((outP1)/50)
SIGNAL = ABSAPRICE(1)
SIGNAL = ABSAPRICE * exp((outP1 + outP2)/50)
SIGNAL = ABSAPRICE(2)

.
.

SIGNAL = ABSAPRICE * exp((outP1 + outP2 +....+ outP10)/50
SIGNAL = ABSAPRICE(10)
END
```

APPENDIX B

ABSA	20050103	7685	7690	7575	796449	7600
ABSA	20050104	7529	7690	7528	1058150	7690
ABSA	20050105	7350	7500	7340	1337221	7490
ABSA	20050106	7169	7360	7150	1917233	7360
ABSA	20050107	7180	7225	7140	1769848	7165
ABSA	20050110	7267	7290	7201	2049115	7201
ABSA	20050111	7260	7420	7250	1978162	7350
ABSA	20050112	7190	7290	7190	1539450	7270
ABSA	20050113	7138	7230	7100	2232427	7200
ABSA	20050114	7199	7277	7100	1902427	7140
ABSA	20050117	7280	7300	7150	989371	7199
ABSA	20050118	7125	7305	7080	2160533	7300
ABSA	20050119	7200	7275	7050	2087389	7125
ABSA	20050120	7201	7281	7170	1912120	7190
ABSA	20050121	7250	7300	7200	1151502	7225
ABSA	20050124	7210	7350	7120	1104477	7150
ABSA	20050125	7280	7300	7200	1234173	7210
ABSA	20050126	7520	7539	7200	2762242	7350
ABSA	20050127	7600	7620	7460	1717287	7500
ABSA	20050128	7616	7665	7505	2318344	7590
ABSA	20050131	7639	7750	7600	1008487	7620
ABSA	20050201	7572	7630	7460	1904946	7615
ABSA	20050202	7460	7645	7460	2873738	7645
ABSA	20050203	7449	7500	7390	1223710	7452
ABSA	20050204	7390	7500	7390	523312	7435
ABSA	20050207	7270	7425	7200	777075	7420
ABSA	20050208	7200	7270	7125	3182200	7270
ABSA	20050209	7430	7430	7182	1458594	7182
ABSA	20050210	7560	7560	7445	1604528	7450
ABSA	20050211	7515	7540	7402	1111606	7402
ABSA	20050214	7540	7550	7480	1681360	7480
ABSA	20050215	7521	7540	7400	652346	7520
ABSA	20050216	7430	7560	7338	957766	7520
ABSA	20050217	7500	7530	7310	989276	7475
ABSA	20050218	7645	7690	7400	732141	7500
ABSA	20050221	7835	7835	7640	975514	7645
ABSA	20050222	7800	7900	7750	652848	7900
ABSA	20050223	7840	7850	7725	766327	7750
ABSA	20050224	7900	7971	7829	1214323	7829
ABSA	20050225	7900	7910	7800	6387185	7800
ABSA	20050228	7949	7950	7800	1783201	7875
ABSA	20050301	8070	8096	7880	1082713	7900
ABSA	20050302	7940	8060	7895	2236940	8060
ABSA	20050303	7921	8000	7770	1925541	7900
ABSA	20050304	7920	7930	7825	625928	7930
ABSA	20050307	8040	8060	7875	598946	7900
ABSA	20050308	8100	8170	8025	1254946	8030
ABSA	20050309	8150	8150	8040	735129	8100
ABSA	20050310	8000	8130	7950	905889	8075
ABSA	20050311	7950	7960	7700	2783135	7950
ABSA	20050314	7891	8000	7850	1153080	8000
ABSA	20050315	7750	7891	7617	1534162	7891

ABSA	20050316	7699	7790	7660	1020014	7790
ABSA	20050317	7750	7779	7601	2293284	7700
ABSA	20050318	7780	7900	7700	984963	7750
ABSA	20050322	7625	7900	7585	1089407	7750
ABSA	20050323	7600	7700	7550	837861	7700
ABSA	20050324	7510	7660	7500	978250	7500
ABSA	20050329	7450	7510	7270	2515840	7510
ABSA	20050330	7400	7550	7400	1208927	7500
ABSA	20050331	7540	7570	7400	1093843	7400
ABSA	20050401	7650	7650	7540	748058	7550
ABSA	20050404	7705	7750	7500	863926	7650
ABSA	20050405	7640	7750	7601	850703	7750
ABSA	20050406	7555	7699	7503	1522291	7640
ABSA	20050407	7550	7660	7455	2072602	7660
ABSA	20050408	7550	7600	7510	1249131	7550
ABSA	20050411	7550	7630	7511	513585	7630
ABSA	20050412	7600	7661	7479	922202	7525
ABSA	20050413	7475	7649	7400	752294	7600
ABSA	20050414	7550	7550	7400	1362047	7400
ABSA	20050415	7820	7820	7450	1507931	7550
ABSA	20050418	7700	7795	7425	913929	7795
ABSA	20050419	7755	7800	7650	545797	7700
ABSA	20050420	7651	7820	7575	761482	7755
ABSA	20050421	7778	7799	7600	1330359	7600
ABSA	20050422	7601	7800	7601	717214	7800
ABSA	20050425	7660	7801	7652	1271611	7801
ABSA	20050426	7820	7820	7640	1189360	7720
ABSA	20050428	7830	7890	7755	2467216	7820
ABSA	20050429	7780	7830	7700	2788754	7825
ABSA	20050503	7770	7810	7700	2146690	7800
ABSA	20050504	7815	7820	7780	1552632	7790
ABSA	20050505	7950	7950	7800	1633948	7850
ABSA	20050506	8000	8101	7950	1506171	7950
ABSA	20050509	8250	8290	8175	5452560	8175
ABSA	20050510	8250	8280	8225	3774470	8280
ABSA	20050511	8260	8300	8230	1443175	8250
ABSA	20050512	8290	8290	8260	548050	8280
ABSA	20050513	8275	8298	8254	921077	8285
ABSA	20050516	8300	8395	8250	1232403	8250
ABSA	20050517	8300	8320	8261	1440280	8300
ABSA	20050518	8251	8300	8250	776435	8300
ABSA	20050519	8250	8300	8051	1222527	8051
ABSA	20050520	8250	8260	8238	210532	8250
ABSA	20050523	8285	8300	8250	858716	8250
ABSA	20050524	8240	8285	8230	953313	8285
ABSA	20050525	8230	8260	8220	408012	8240
ABSA	20050526	8200	8270	8190	1278120	8245
ABSA	20050527	8175	8220	8145	2124685	8200
ABSA	20050530	8220	8275	8150	866178	8275
ABSA	20050531	8244	8270	8200	1674971	8200
ABSA	20050601	8240	8260	8200	1445523	8260
ABSA	20050602	8250	8260	8230	1808826	8240
ABSA	20050603	8261	8281	8250	563520	8250
ABSA	20050606	8290	8340	8242	581307	8242
ABSA	20050607	8345	8380	8265	1079557	8325

ABSA	20050608	8300	8375	8300	1123356	8300
ABSA	20050609	8300	8340	8262	646808	8262
ABSA	20050610	8275	8350	8275	1348922	8300
ABSA	20050613	8290	8335	8271	912573	8271
ABSA	20050614	8310	8310	8290	995516	8310
ABSA	20050615	8338	8350	8285	1258004	8325
ABSA	20050617	8340	8374	8310	1576316	8310
ABSA	20050620	8155	8239	8141	691890	8152
ABSA	20050621	8175	8180	8165	1073528	8167
ABSA	20050622	8170	8195	8150	1800026	8150
ABSA	20050623	8229	8230	8170	2905196	8170
ABSA	20050624	8190	8235	8155	1840981	8235
ABSA	20050627	8187	8190	8160	2111811	8190
ABSA	20050628	8203	8215	8175	2751761	8187
ABSA	20050629	8260	8275	8201	2238163	8210
ABSA	20050630	8248	8260	8230	637325	8260
ABSA	20050701	8251	8275	8230	2006394	8250
ABSA	20050704	8250	8300	8240	527778	8240
ABSA	20050705	8250	8265	8230	2691347	8250
ABSA	20050706	8245	8255	8240	4366975	8250
ABSA	20050707	8255	8265	8242	2194166	8250
ABSA	20050708	8254	8255	8249	1596420	8250
ABSA	20050711	8251	8275	8250	1742397	8250
ABSA	20050712	8300	8300	8250	1727593	8250
ABSA	20050713	8420	8420	8280	7661309	8285
ABSA	20050714	8761	8761	8420	4680387	8420
ABSA	20050715	8595	8820	8505	3275455	8700
ABSA	20050718	8700	8770	8600	2245587	8600
ABSA	20050719	8970	8970	8700	1459396	8701
ABSA	20050720	9100	9220	8900	3308708	8950
ABSA	20050721	9050	9140	9000	1047359	9100
ABSA	20050722	9050	9078	9010	465068	9055
ABSA	20050725	9275	9345	8801	623483	9050
ABSA	20050726	9299	9495	9210	3292742	9231
ABSA	20050727	9375	9410	9051	1742356	9200
ABSA	20050728	9400	9480	9360	2357948	9400
ABSA	20050729	9155	9500	9155	1659924	9400
ABSA	20050801	9316	9350	9050	15601554	9101
ABSA	20050802	9360	9451	9200	3005013	9320
ABSA	20050803	9370	9395	9260	615787	9345
ABSA	20050804	9475	9500	9300	842912	9300
ABSA	20050805	9571	9590	9475	1708436	9475
ABSA	20050808	10000	10000	9520	3419081	9570
ABSA	20050810	9915	10000	9850	3369657	9900
ABSA	20050811	9850	9950	9700	2984985	9916
ABSA	20050812	9701	9820	9610	3222144	9765
ABSA	20050815	9530	9700	9500	1822879	9700
ABSA	20050816	9275	9590	9205	1221582	9590
ABSA	20050817	9199	9300	9050	3532671	9275
ABSA	20050818	9180	9295	9135	3786688	9135
ABSA	20050819	9369	9400	9150	1960781	9165
ABSA	20050822	9496	9496	9387	794920	9420
ABSA	20050823	9500	9550	9460	1107673	9500
ABSA	20050824	9485	9540	9450	1153706	9500
ABSA	20050825	9460	9500	9280	897035	9400
ABSA	20050826	9512	9575	9415	1085904	9480

ABSA	20050829	9474	9650	9400	521913	9450
ABSA	20050830	9300	9474	9300	483977	9450
ABSA	20050831	9210	9320	9150	1213183	9320
ABSA	20050901	9245	9400	9175	882189	9210
ABSA	20050902	9480	9550	9255	333844	9255
ABSA	20050905	9700	9727	9460	583531	9460
ABSA	20050906	9725	9977	9610	603446	9700
ABSA	20050907	9825	9995	9700	2299104	9700
ABSA	20050908	9800	9900	9695	975632	9825
ABSA	20050909	9660	9870	9520	556631	9520
ABSA	20050912	9640	9750	9520	1197395	9660
ABSA	20050913	9480	9610	9369	1671324	9600
ABSA	20050914	9450	9500	9340	1531292	9400
ABSA	20050915	9325	9540	9255	2731991	9450
ABSA	20050916	9205	9400	9200	1812465	9325
ABSA	20050919	9140	9300	9120	5000884	9215
ABSA	20050920	9200	9280	9140	1916449	9140
ABSA	20050921	9200	9300	9200	2260420	9220
ABSA	20050922	9041	9200	9005	1240598	9200
ABSA	20050923	9000	9110	8800	1035659	9005
ABSA	20050926	9240	9270	9010	1008511	9010
ABSA	20050927	9220	9300	9100	948030	9300
ABSA	20050928	9270	9299	9200	2242149	9280
ABSA	20050929	9425	9425	9270	1244559	9270
ABSA	20050930	9380	9480	9235	1007516	9425
ABSA	20051003	9100	9360	9010	618849	9300
ABSA	20051004	8985	9121	8915	627444	9120
ABSA	20051005	9120	9170	8900	1325767	8900
ABSA	20051006	8950	9095	8900	479905	9020
ABSA	20051007	8900	9050	8700	2117218	8950
ABSA	20051010	8745	8955	8745	1037688	8950
ABSA	20051011	8840	8920	8700	1875231	8745
ABSA	20051012	8835	8980	8800	1535815	8854
ABSA	20051013	8660	8820	8448	1700039	8800
ABSA	20051014	8465	8700	8411	1200637	8665
ABSA	20051017	8770	8770	8514	1667155	8650
ABSA	20051018	8720	8980	8660	1051700	8775
ABSA	20051019	8525	8725	8510	2305456	8700
ABSA	20051020	8570	8625	8500	712301	8625
ABSA	20051021	8500	8600	8400	568846	8400
ABSA	20051024	8614	8750	8512	902092	8550
ABSA	20051025	8640	8800	8620	1438372	8700
ABSA	20051026	8799	8800	8625	855457	8650
ABSA	20051027	8600	8799	8600	377881	8799
ABSA	20051028	8800	8800	8525	641331	8600
ABSA	20051031	8900	8985	8760	799215	8830
ABSA	20051101	8888	8950	8750	883022	8900
ABSA	20051102	8915	9050	8650	1560410	8880
ABSA	20051103	9020	9070	8950	1126428	9070
ABSA	20051104	9026	9140	8975	503040	9070
ABSA	20051107	9230	9230	8901	343126	9020
ABSA	20051108	9000	9230	9000	1198408	9230
ABSA	20051109	8925	9050	8850	491358	9000
ABSA	20051110	9000	9000	8700	1048567	8700
ABSA	20051111	9000	9292	9000	1692280	9070
ABSA	20051114	9120	9200	9000	1472289	9200

ABSA	20051115	9085	9145	9025	622473	9120
ABSA	20051116	8920	9100	8801	620622	9080
ABSA	20051117	8950	9085	8910	1217715	8975
ABSA	20051118	8930	9000	8902	1318012	8985
ABSA	20051121	8880	9030	8876	1063933	9000
ABSA	20051122	8900	8940	8800	2152962	8901
ABSA	20051123	9227	9227	8901	2731165	8901
ABSA	20051124	9318	9375	9176	1321962	9227
ABSA	20051125	9330	9370	9250	900755	9250
ABSA	20051128	9300	9429	9255	592072	9380
ABSA	20051129	9200	9300	9160	425487	9300
ABSA	20051130	9180	9340	9025	1152907	9200
ABSA	20051201	9050	9360	9050	1893012	9060
ABSA	20051202	8900	9199	8900	2355374	9060
ABSA	20051205	9036	9049	8841	1081357	8980
ABSA	20051206	9285	9380	9036	1594838	9036
ABSA	20051207	9718	9718	9280	1105718	9300
ABSA	20051208	9625	9670	9400	1366308	9600
ABSA	20051209	9810	9810	9550	2850406	9550
ABSA	20051212	10000	10100	9700	3762940	9800
ABSA	20051213	9930	10050	9860	1715207	9998
ABSA	20051214	9950	10050	9780	1091413	9900
ABSA	20051215	9970	10000	9901	4181228	10000
ABSA	20051219	9900	9949	9821	1732209	9900
ABSA	20051220	9780	9890	9610	1412119	9610
ABSA	20051221	9900	9900	9720	2320883	9720
ABSA	20051222	10195	10205	9850	1584652	9895
ABSA	20051223	10170	10251	10101	163297	10185
ABSA	20051227	10320	10320	10170	231635	10170
ABSA	20051228	10130	10300	10061	206505	10300
ABSA	20051229	10010	10200	9990	161272	10130
ABSA	20051230	10100	10110	9990	255521	10060

Bibliography

Books:

1. **[Ami92]** D. Amit, "Modelling Brain Function", Cambridge University Press, 1992,
2. **[Arn02]** G. Arnold, "Valuegrowth Investing", Pearson Education Ltd., 2002,
3. **[Azo94]** M. Azoff, "Neural Network Time Series Forecasting of Financial Markets", John Wiley & Sons, 1994,
4. **[DBH05]** M. Hagan, H. Demuth, M. Beale "Neural Network Design", Campus Publishing Service, University of Colorado at Boulder, 2005,
5. **[Dim88]** E. Dimson, "Stock Market Anomalies" Cambridge University Press, 1988,
6. **[DN91]** E. Davalo, P. Naim, "Neural Networks", Macmillan Education Ltd., 1991,
7. **[GNZ02]** H.G. Zimmermann, R. Neuneier, R. Grothmann, "Modelling dynamical Systems by Error Correction", Eds. Proceedings IDEAL, 2002,
8. **[Hul03]** J. Hull, "Options, Futures and other Derivatives", Prentice Hall, 2003,
9. **[Kea83]** S. Keane, "Stock Market Efficiency" Phillip Allan Publishers Ltd., 1983,
10. **[Liv99]** L. Livens, "Share and Business Valuation Handbook", Tolley Publishing, 1999,
11. **[Mcn05]** P. McNelis, "Neural Networks in Finance", Elsevier Academic Press, 2005,
12. **[MD41]** N. Miller, J. Dollard, "Social Learning and Imitation", Kegan Paul, Trench, Trubner & Co Ltd 1941,
13. **[Mos01]** S. Moscovici, "Social Representations", New York University Press, 2001,
14. **[NZ98]** R. Neuneier, H.G. Zimmermann, "How to train Neural Networks", Springer, 1998,
15. **[NZ00]** H.G. Zimmermann, R. Neuneier, „Modeling Dynamical Systems by Recurrent Neural Networks", WIT Press, 2000,
16. **[NYG04]** K. Nygren, "Stock Predicition – A Neural Network Approach", Master Thesis at the Royal Institute of Technology, KTH, 2004,
17. **[Plu03]** T. Plummer, "Forecasting Financial Markets", Kogan Page Inc. 2003,
18. **[She83]** I. Sherman, "Biology, a Human Approach", Oxford University Press, 1983,
19. **[Tar98]** L. Tarassenko, "A Guide to Neural Computing Applications", John Wiley & Sons Inc., 1998,
20. **[Wat80]** G. Watson, "Approximation Theory and Numerical Methods", John Wiles & Sons, 1980,
21. **[Wär02]** K.-E. Wärneryd, "Stock-Market Psychology. How People value and trade stocks", Edward Elgar Publishing Inc., 2002,
22. **[Zim94]** H.G. Zimmermann, "Neuronale Netze als Entscheidungskalkül", Franz Vahlen, 1994,
23. **[Zim05]** H.G Zimmermann, "System Identification and Forecasting", Siemens AG. CT IC 4, 2005,
24. **[Zir97]** J. Zirilli, "Financial Prediction using Neural Networks", International Thomson Publishing Company, 1997

Web Information:

25. **[AEHP05]** R. Anderson, K. Eom, S. Hahn, J.-H. Park, "Stock Return Autocorrelation is Not Spurious" a working paper from the departments of Economics and Mathematics, University of California at Berkley, 2005,
26. **[BC97]** H. Bessembinder, K. Chan, "Market Efficiency and the Returns to Technical Analysis" in a working paper from the Department of Finance, Arizona State University, December 1997,
28. **[Bra]** www.demo.cs.brandeis.edu/pr/DIBA

29. **[Fri06]** C. Fries, "Finanzmatematik, Theory, Modellierung und Implementierung", online beta version, 2006,

30. **[Gro02]** R. Grothmann, "Multi-Agent Market Modeling based on Neural Networks", E-Lib, 2002,

31. **[Har04]** www.cedar.buffalo.edu/~srihari/CSE574/Cha4.Part2.pdf,

32. **[Idc06]** www.investorsdictionary.com,

33. **[Jes]** J. Jesan. "The Neural Approach to Pattern Recognition". www.acm.org/ubiquity/views/v5i7_jesan.html,

34. **[Sc06]** http//stockcharts.com/school/doku.php?id=chart_school:chart_analysis:chart_patterns

35. **[SS96]** C. Stergiou, D. Siganos, "Neural Networks", university working paper under www.doc.ic.ac.uk/~nd/surprise_96/journal/vol4/cs11/report.html

36. **[Sza]** J. Szabados, "Discrete linear interpolatory Operators" from http://www.emis.de/journals/SAT/papers/5/5.pdf

Journals:

37. **[BS73]** F. Black, M. Scholes, "The Pricing of Options and Corporate Liabilities" in the Journal of Political Economy, Vol.81(3), 1973,

38. **[Bla86]** F. Black "Noise" in the Journal of Finance Vol.41(3), July 1986,

39. **[DT98]** W. De Bondt, R. Thaler, "A Portrait of the Individual Investor" in the European Economic Review Vol. 42, 1998,

40. **[Ell01]** C. Ellwood "The Theory of Imitation in Social Psychology" in the Journal of Sociology Vol.6,

41. **[Fam65]** E. Fama, "The Behaviour of Stock Market Prices" in the Journal of Business Vol. 38(1), January 1965,

42. **[Fam70]** E. Fama, "Efficient Capital Markets: A Review of Theory and Empirical Work" in the Journal of Finance Vol. 25(2), May 1970,

43. **[Fam91]** E. Fama, "Efficient Capital Markets II" in the Journal of Finance Vol. 46(5), December 1991,

44. **[GER95]** H. Geman, N. El Karoui, J.-C. Rochet, „Changes of Numeraire, Changes of Probability Measure And Option Pricing" in the Journal of Applied Probability, Vol.32, 1995,

45. **[Jen78]** M. Jensen, "Some Anomalous Evidence Regarding Market Efficiency" in the Journal of Financial Economics Vol. 6, 1978,

46. **[JT93]** N. Jegadeesh, S. Titman, "Returns to Buying Winners And Selling Losers: Implications for Stock Market Efficiency" in the Journal of Finance Vol. 48(1), March 1993,

45. **[LM88]** A. Lo, C. MacKinlay, "Stock Market Prices Do Not Follow Random Walks: Evidence from a Simple Specification Test" in the Review of Financial Studies Vol.1(1), 1988,

46. **[LP81]** S. LeRoy, R. Porter, "The Present Value Relation: Tests based on Implied Variance Bounds" in Econometrica Vol. 49(3), May 1981,

47. **[Mal03]** B. Malkiel, "The Efficient Market Hypothesis and its Critics" in the Journal of Economic Perspectives Vol.17(1), December 2003,

48. **[PTY99]** J. Yao, C.L. Tan, H.-L. Poh, "Neural Networks for technical analysis: A study on KLCI" from the international Journal of theoretical and applied Finance Vol.2(2), 1999,

49. **[Rob59]** H. Roberts, "Stock Market Patterns and Financial Analysis: Methodological Suggestions" in the Journal of Finance Vol. 14(1), March 1959,

51. **[Rou98]** G. Rouwenhorst, "International Momentum Strategies" in the Journal of Finance Vol.53(1), February 1998.

52. **[SHI81]** R. Shiller, "Do Stock Prices Move Too Much to be Justified by Subsequent Changes in Dividends" in the American Economic Review Vol.71(3), January 1981.

53. **[TG04]** A. Timmermann, C. Granger "Efficient market hypothesis and forecasting" in the international Journal of Forecasting Vol.20, 2004.