NTNU
Norwegian University of
Science and Technology

Using Artificial Neural Networks To Forecast Financial Time Series

Rune Aamodt

Master of Science in Computer Science
Submission date: June 2010
Supervisor: Arvid Holme, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Problem Description

The student will investigate how artificial neural networks can be trained to forecast developments of financial time series.
He will first need to establish whether any similar research has been conducted previously, and if so to review the various approaches to the problem suggested therein.
Following this prestudy, the student should decide on an approach and make the necessary implementations to train and test the neural networks. The attainable forecasting performance should be evaluated emprically.
Simulations will be done using historical intraday trade data which has been procured for a selection of stocks from the Oslo Stock Exchange.

Assignment given: 15. January 2010
Supervisor: Arvid Holme, IDI

Abstract

This thesis investigates the application of artificial neural networks (ANNs) for forecasting financial time series (e.g. stock prices).

The theory of technical analysis dictates that there are repeating patterns that occur in the historic prices of stocks, and that identifying these patterns can be of help in forecasting future price developments. A system was therefore developed which contains several "agents", each producing recommendations on the stock price based on some aspect of technical analysis theory. It was then tested if ANNs, using these recommendations as inputs, could be trained to forecast stock price fluctuations with some degree of precision and reliability.

The predictions of the ANNs were evaluated by calculating the Pearson correlation between the predicted and actual price changes, and the "hit rate" (how often the predicted and the actual change had the same sign). Although somewhat mixed overall, the empirical results seem to indicate that at least some of the ANNs were able to learn enough useful features to have significant predictive power.

Tests were performed with ANNs forecasting over different time frames, including intraday. The predictive performance was seen to decline on the shorter time scales.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter summarizes the basic motivation of this thesis and gives a brief overview of the contents in this report.

1.1 Motivation

The main purpose of the work presented in this report is to investigate if and how artificial neural networks (ANNs) can be used to forecast financial time series (i.e. the price curve of financial securities). As we will see in chapter 4, several researchers have already performed similar investigations, however there are some novel features of the approach used in this thesis that separates it from the bulk of the existing research.

Probably the most important of these differences is that the empirical tests in this thesis were performed with intraday trade data, whereas the previous research has generally been carried out with only daily data (i.e. one data value for each day). So while the existing research has been restricted to mid-term and long-term forecasting, this thesis is unique in that it also investigates the viability of applying ANNs to short-term intraday forecasting.

Another major difference between this thesis and most other research is that the forecasting models tested here utilize heuristic methods inspired

by the discipline of technical chart analysis (chapter 3) in an effort to help the ANNs extrapolate relevant features of the data. The vast majority of the existing research is not based on any such method; simply applying the ANNs to raw price data seems to be the norm. This is discussed at greater length in chapter 4.

1.2 Krang

The Krang system is an application that was developed to carry out the empirical studies in this thesis. Its functionality, which includes the creation, training and evaluation of forecasting ANNs with intraday stock price data, is described with great detail in chapter 5.

1.3 Report Structure

This report can be seen as having three major parts: Chapters 2-4 summarize what was found during the prestudy phase of the thesis work. Chapters 5-6 describe the functionality of the Krang system, and exactly how it was used to generate the empirical results of this thesis. Chapters 7-9 list these results, along with some commentary/discussion leading up to the final conclusion.

As for the contents of the prestudy, chapter 2 provides some background perspective on financial markets in general. Chapter 3 introduces some of the concepts of technical analysis, with emphasis on the parts that are relevant for the Krang system. Chapter 4 provides a brief introduction to what artificial neural networks are, and reviews some of the existing research where ANNs have been used to forecast financial markets.

Chapter 2

Financial Markets

Since this report assumes no prior knowledge of finance on the part of the reader, it seems appropriate to provide an overview of some of the basic theory. This chapter explains what financial markets are and how they work. Some key financial concepts are also explained which are relevant to the rest of this report.

2.1 Financial Securities

A financial security, or financial asset, is basically a marketable contract that represents a claim on some present or future value. Broadly speaking, there are three types of securities:[28]

Equity (i.e. stocks) denote part ownership of a business. Each share of stock typically grants the owner voting rights when stockholders vote on company decision. It also entitles the holder to any cash dividends the company might decide pay to its stockholders from their profits.

Debt securities grant the holder rights to some future cash payments from the issuer of the contract. One example is government bonds, which are issued by governments when they need to borrow money.

Derivatives are contracts whose present value is tied to the future value of some other underlying asset. A common example of such a contract is the stock option. This contract pays a sum of money on a given date based on how much the price of a stock is above or below a given level on that date.

2.2 Background

Now that we have a basic understanding of what financial securities are, we can consider the context in which they are bought and sold. In particular, some perspective may be gained from considering how these financial markets are organized and how they came to be that way. We will also explain the practice of algorithmic trading, where computers programs themselves become autonomous market participants.

2.2.1 The Evolution of Financial Markets

The earliest known trading of financial securities dates back to several thousand years B.C. when Sumerians would organize auctions for primitive commodity futures contracts made out of clay (a futures contract is a type of derivative that would allow the issuing farmer to secure a price for the future sale of his crops ahead of harvest). [11][26]

In the western world, financial markets have undergone a tremendous evolution over the past few centuries. The worlds first official stock exchange was opened in Amsterdam in 1602[24], and by the middle of the 19th century there were a large number of exchanges operating all over the western world. The existence of stock exchanges were vital to the industrial growth of the world during the 18th and 19th centuries, as they provided companies with a pool of capital to which they could sell their own shares in order to fund business expansion.[27]

Until the 1960s, stock exchanges (and financial markets in general) were organized as physical locations where brokers would meet and exchange buy/sell orders in an open outcry auction. But with the advent of digital

communication technology, trading quickly became more and more computer driven, which allowed traders in remote locations to send their orders electronically to the exchange.

Traditional auction trading still takes place in various locations, as can be seen in fig. 2.1. This is a photo from a trading pit in Chicago where traders still trade commodity futures contracts in person. In todays world, however, the vast majority of financial trading is purely electronical. The work environment of a modern day trader is more likely to look something like fig. 2.2.

One of the largest stock exchanges in the world, the NASDAQ, is managed completely electronically.[2, p.5] The same is true for the Norwegian stock exchange, the Oslo Stock Exchange (OSE), which closed down its physical stock trading pits in 1999 when it switched to an all-electronic system.[14]

Financial markets exist for all the previously mentioned types of financial securities: stocks, bonds and derivatives. In addition to these, there is also the FX in which currencies are traded. The FX are generally the most liquid financial markets in the world, totaling more than 4 trillion USD in trading volume on an average day.[7]

Figure 2.1: A trading pit in the Chicago Mercantile Exchange (CME)

Figure 2.2: An example of a modern traders work desk

2.2.2 Algorithmic Trading

As the financial markets became increasingly tech driven, several investors and institutions realized that there was a potential for automatic computerized trading systems which could trade securities without human input, so called algorithmic trading systems.

The earliest such systems would only try to scalp small profits by looking for arbitrage opportunities (an arbitrage opportunity occurs when something can be bought and sold at a profit instantaneouly). Consider for example three FX markets: USD/GBP, GBP/EUR and USD/EUR. Situations could occur where these currency pairs were being traded with a small disrepancy so that buying a dollar in pounds, selling the dollar in euros and buying back pounds with the resulting euros would result in a small (yet immediate) profit. Such opportunities soon became scarce as more and more algorithmic systems started competing to find them, driving tech savvy investors to look for more creative approaches to algorithmic trading.

As computer technology became more and more pervasive, the algorithmic trading systems also became more sophisticated. By the late 1980s complex automated trading systems were already a common occurence in the US markets, especially for rich institutions and hedge funds. When the worlds stock markets fell by record amounts on October 19, 1987 ("Black Monday"), many blamed automated trading systems for exacerbating the decline as they supposedly started blindly selling stocks.[25] Whether these allegations are true remains speculation to this day, however the mere fact that they were concieved does indicate that such systems were already common at that time.

The Black Monday incident is not the only event with which algorithmic trading has been painted a villain. So called *high frequency trading* (HFT) systems, which are a special class of algorithmic trading systems, have recently been accused of manipulating markets and having unfair advantages over common investors. Some sources estimate that HFT systems presently account for over 70% of stock trading volume in the U.S.[3] These systems can make hundreds of thousands of quick trades every day, scalping small

profits of short term movements in the prices of securities.

From a research perspective, the main problem with these systems is that although they are clearly being used pervasively by certain institutions, the specific details of how they are implemented and their actual profitability when deployed in practice has never been published for the scientific community at large. As we shall see in chapter 4, some (public) research has been conducted in applying various machine learning techniques into financial trading, but the literature is scarce and the approaches that are tested are often somewhat simplistic (compared to what you might expect an investment firm to develop proprietarily).

2.3 Financial Time Series

In general, a *time series* refers to a series of data points which are measured at successive points in time spaced at uniform time intervals. This concept is heavily used in scientific fields like statistics and signal processing, but also in the context of financial analysis.[29]

The price that a particular security is traded at can be viewed as a time series, where the value at a given point in time is the price of the last observed trade at that time. In essence, the time series is then just a simple price curve for the security. An example is given in fig. 2.3 where the time series for Google stock is plotted spanning the year 2009. In this particular plot, the sampling interval is once per day, and so each value in the plot is the last traded price (or *closing* price) of that day.

2.3.1 Candlestick Time Series

One drawback of the time series as it was just defined is that it may give an incomplete picture of the volatility in the price of the underlying security. This is because each point on the curve only displays a single price value, and says nothing about whether the price fluctuated in that interval or not. To get a more complete sense of the price movements for the security in each time interval, we might instead choose to use the so-called candlestick series.

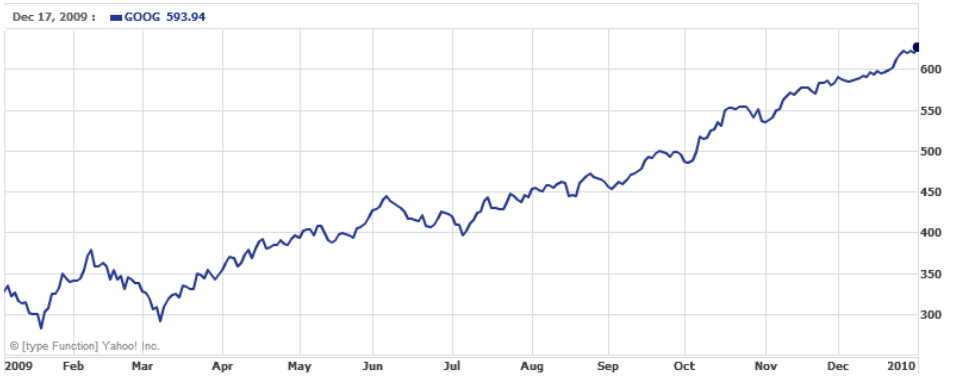

Figure 2.3: Time series (line chart) for Google stock over the year 2009 (source: Yahoo! Finance[31])

An example of a candlestick series is given in fig. 2.4. Here we see that instead of just singular values, each point in the series is represented by a colored bar and line. Each of these items is what we call a candlestick, or just candle for short. The graphical features of a candlestick can be interpreted as follows:

- The thick rectangular bar stretches from the price of the first trade in the candle (the *open*) to the price of the last trade (the *close*).
- The thin spine of the candle stretches from the highest price of any trade in the candle (the *high*) to the lowest price (the *low*).
- A green colored candle means that the closing price was higher than the open (i.e. the stock price went up in the time period represented by the candle). A red colored candle means the opposite (i.e. the price went down).
- If no trades occured during the time period of the candle, it is simply displayed as a flat line at the same level as the close of the previous candle.

Figure 2.4: Candlestick chart for Google stock over the first few months of 2009 (source: Yahoo! Finance[31])

2.4 The Efficient Market Hypothesis

The *efficient market hypothesis* (EMH) is a theory which states that all information about a security is already taken into account by its market price. The main consequence of this is that it should be impossible to "outsmart" the overall market.

The theory is based on the assumption that all relevant information is publicly available and easily accessible to all investors, and that investors act rationally. If these assumptions hold, competition among investors should spontaneously and immediately negate any speculative profit opportunities from the moment new information becomes available.

A natural consequence of the EMH is that financial time series are always unpredictable. To further formulate this assertion using statistics, the *random walk* hypothesis was proposed by Paul Cootner in 1964[4]. This theory states that all financial time series are statistically equivalent to a series of completely random steps, and that attempting to make predictions based on historical analysis is always a futile endeavor.

If this were true, then it should in general be impossible for algorithmic trading systems to be more profitable (at least on a risk-adjusted basis)

than the overall market. It also flies in the face of technical analysis (described in chapter 3) which is based entirely on historical analysis and the assumption that repeating patterns occur in financial time series that allow for profitable trading opportunities.

There has been a lot of criticism against the EMH. For one, it assumes that investors are always rational, but many behavioral economists argue that the presence of cognitive biases (such as overconfidence and overreaction) negate the validity of this assumption. Another problem is that it does not take into account the presence of insider trading, where small groups of investors trade based on information which is not publicly available.

Many researchers have attempted to produce falsifications of the EMH and the random walk hypothesis, with mixed results. One of the most commonly cited of these efforts is that of Andrew W. Lo and Archie C. MacKinlay, who in their book *A Non-Random Walk Down Wall Street* developed a statistical model for analysing stock prices which they claim provided significant evidence against random walk theory based on its empirical results.[13]

2.5 Trading Nomenclature

In addition to what we have introduced so far in this chapter, there are certain terms that exist among traders which are also used in the later chapters of this report. This section gives a brief review of the terminology that it is recommended for the reader to be familiar with in order to comprehend everything in this report.

2.5.1 The Bull And The Bear

In the context of financial trading, the image of the bull is used allegoricaly to describe the belief that the price of a stock (or all stocks) is going to increase in the future. A trader who holds such a belief is said to have a bullish sentiment.

To represent the opposite belief, namely that the price is going to decline, the image of the bear is invoked.

2.5.2 Long Or Short

The terms long and short are used in a similar manner to the bull and the bear, but in somewhat different contexts. The former term, long, is rather straightforward to understand: An investor who has bought and owns some positive amount of a given security, is said to have a long position in that security.

A short position is in principle just as simple, as it is just the opposite of a long position. A short sale occurs when an investor sells a security he does not own, meaning he has to borrow it from someone else in order to sell it. After the sale has taken place, he now owns a negative amount of the security (i.e. a short position) and will profit only if the price of the security goes down and he covers his short position by buying the same amount he sold for the now lower price.

So by allowing both short and long positions, a trader can make money whether the stock goes down or up, as long as he can predict the direction of the movement in advance. Short positions can be very dangerous though; since the price of the stock can climb by more than 100%, the trader may have to pay more than double what the loan was worth originally to cover his short. For a long position, the worst case scenario is that the security becomes worthless which means a loss of 100%. For this reason, short positions are rarely good long term bets and are mostly used for short term speculation.

Figure 2.5: The bull statue on Wall Street symbolizes a positive market sentiment

Chapter 3

Technical Analysis

Technical analysis is a discipline for analyzing the historical trading activity of financial securities (with the aim of forecasting future price movements). It comprises a set of techniques which look for specific reoccuring patterns in the price charts of the security in order to identify trend formations and other repeating patterns that may occur.

In this chapter a brief review of technical analysis theory is provided, with a primary focus on the parts of the theory which are relevant to the rest of this report.

3.1 Swing Points

In the price history of a security, we refer to the peaks and troughs that occur in the price over time as the swing points of the time series. In technical analysis, these points play a key role in the identification of support and reistance levels on the price curve. Support means a price level at which we expect the demand of the security to increase relative to the supply. Resistance means the opposite; a price level where we expect the supply to increase relative to the demand.

The simplest way to determine potential support and resistance levels using the swing points on a chart is simply to extend a horizontal line from

Figure 3.1: Using swing points to identify support and resistance levels (source: *StockCharts.com*[20])

a swing point. This method is illustrated in fig. 3.1, where we see that the first (low) swing point determines a support level which the stock price later retracts towards several times but fails to pierce. We also see that the subsequent (high) swing point determines a resistance level which acts as an imaginary roof for the successive price movements.

3.1.1 Trendlines

The swing points can also be used as a basis for identifying trendlines. Instead of just drawing a horizontal line out from each point, we can connect two or more consecutive troughs (or peaks) and extend a line through both. This forms a potential trendline, which we can expect the price curve to encounter support or resistance when it approahces.

Figure 3.2: Example of a support trendline (source: *StockCharts.com*[20])

An example of a trendline is given in fig. 3.2.

3.2 Moving Averages

One of the most commonly referenced technical analysis indicators is the *moving average* of the share price. The basic idea of this indicator is to get a smoothed version of the price curve by taking the average price of the previous N periods at every point.

A moving average can be calculated in several different ways. Let's have a look at two of the more common varieties in detail before we proceed with our discussion.

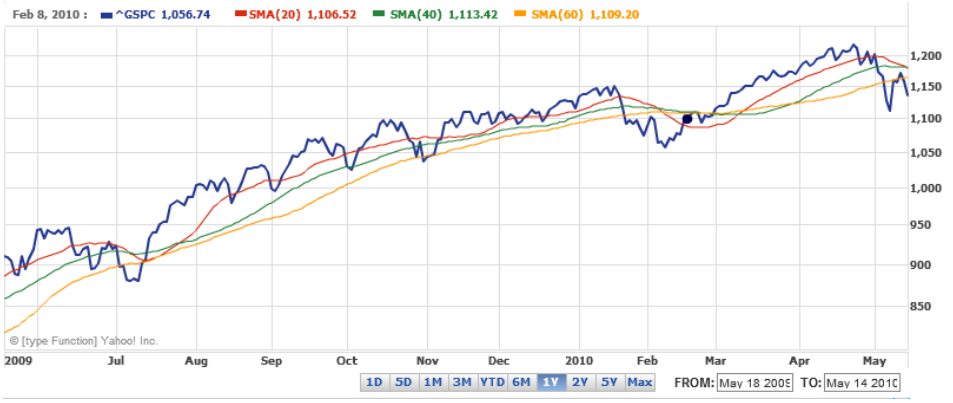

Figure 3.3: Comparison of several simple moving averages on the S&P 500 index curve (source: *Yahoo! Finance*[31])

3.2.1 Simple Moving Average

The most common type of moving average is the *simple moving average* (SMA). An N-period SMA can be calculated at any point in time by taking the mean price of the preceeding N periods:[20]

$$SMA_N(t) = \frac{1}{N} \sum_{i=0}^{N-1} p(t-i) \tag{3.1}$$

An example of three different SMA series are given in fig. 3.3. This chart shows the history of the S&P-500 index over a 1-year period with three moving averages drawn over: the 20-day (red), the 40-day (green) and the 60-day (orange). As you can see, the higher the N parameter (i.e. days) is set, the smoother the curve becomes, and the more its momentum lags that of the underlying curve.

3.2.2 Exponential Moving Average

The *exponential moving average* (EMA) is calculated with the following recursive relation:[16]

$$EMA_N(t) = \frac{1}{N} \times p(t) + \left(1 - \frac{1}{N}\right) \times EMA_N(t-1) \tag{3.2}$$

The first EMA value of the series is simply set to the price at that time, i.e.:

$$EMA_N(t=0) = p(t=0)$$

The EMA can be thought of as an approximation of the SMA that is somewhat easier to compute. But it has an interesting property in that it can be viewed as a weighted average in which the more recent values are weighted with geometric proportion over the older values.[16]

Figure 3.4: Comparison of simple and exponential moving averages on the S&P 500 index curve (source: *Yahoo! Finance*[31])

Fig. 3.4 compares a 30-day SMA with a 30-day EMA on the S&P-500 index curve. We can see that the EMA curve seems to be somewhat more volatile because it swings out further than the SMA in both directions. The EMA also seems to be less lagging than the SMA, if we take a look at

how long it takes before their respective directions change after a peak or trough in the underlying curve. This makes sense in light of the remarks about how the EMA can be thought of as a weighted average, since the fact that later values carry more influence implies that the curve should be more responsive to change.

In the end, we still see that these two moving average varieties are indeed heavily correlated and almost the same for the most part. This is just as we would expect based on the inherent similarities of the calculations.

3.2.3 Analysing The Moving Average

The moving average can be used for making different types of observations when analysing the curve. The two most common are:[20]

- To determine wether the curve is in an upward or downward trend.
- To determine if the curve is near a support or resistance level.

The most obvious type of observation we can make is to use moving averages to classify whether the market is in an upward or downward trend. This can be done simply by classifying the current trend as an upward trend if the current price is higher than the moving average, or oppositely a downward trend if the price lower. This type of classification is illustrated in fig. 3.5, where the areas of transition between trend directions are highlighted for clarification.

A variation of this type of trend classification, is to use two moving averages, one with a lower and one with a higher number of periods N. We then classify the trend as downward if the lower-N moving average is below the higher-N moving average and vice versa for an upward trend.

Another use of moving averages is in looking for support and resistance levels on the curve. In this analysis, we expect an upward trend to meet support as it approaches the moving average from above (for a so-called "bounce"). In the case of a downward trend, we expect it to meet resistance as it approaches the moving average from below. This type of classification is illustrated in fig. 3.6, where the green arrows indicate that the moving average gives support and the red arrows indicate resistance.

Figure 3.5: Using a moving average for trend classification (source: *StockCharts.com*[20])

3.3 Trend Reversal Patterns

Technical analysis dictates that there are specific patterns which often occur at the end of significant trends. The occurence of these patterns can be interpreted as a signal that the stock may be headed into a reverse trend.

When identifying reversal patterns, it's always important to establish that the stock is coming out of a prior trend (be it an uptrend or downtrend).

There are several types of reversal patterns. Our discussion shall be limited to only three of these; the head and shoulders, the double top and the double bottom.

Figure 3.6: Using a moving average to find support and resistance levels (source: *StockCharts.com*[20])

3.3.1 Head And Shoulders Pattern

The most classic reversal pattern is the *head and shoulders* (HS) pattern. An example of the HS pattern is given in fig. 3.7.

The HS is formed by having a high peak ("head") surrounded by two lower peaks ("shoulders"). The *neckline* of the HS is formed by drawing a line between the two troughs that separate the head from the shoulders. When the stock goes beneath this line (after the right shoulder) technical analysis dictates that we should expect the price to fall to the same distance below this line as the distance from the head to the line.[20]

A reverse HS pattern can occur at the end of a downtrend. The same principles apply for this pattern, except that the head and shoulders are marked by troughs in the curve instead of peaks.

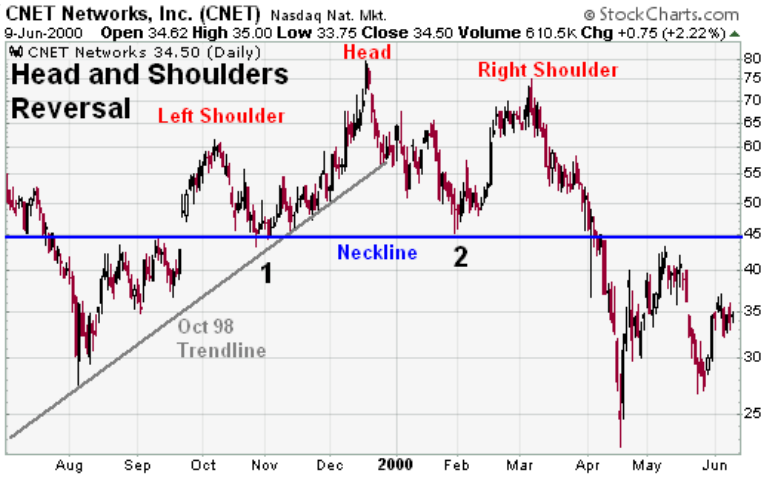

Figure 3.7: Example of the head and shoulder reversal pattern (source: *StockCharts.com*[20])

3.3.2 Double Top Pattern

Another important reversal pattern is the double top. This pattern is relatively simple to recognize, with two peaks occuring consecutively at approximately the same price level. An example of a double top pattern is given in fig. 3.8.

As the curve retreats from the second peak, we should look for an increase in volume to further confirm that the pattern is forming. As the price breaks out below the bottom of the double top channel, we can calculate the expected value of the further decline as the level difference between the peaks and the intermediary trough subtracted from price at the bottom of the channel.[20]

3.3.3 Double Bottom Pattern

The double bottom pattern has the same properties as the double top, except that it happens after a downtrend and that we should look for two

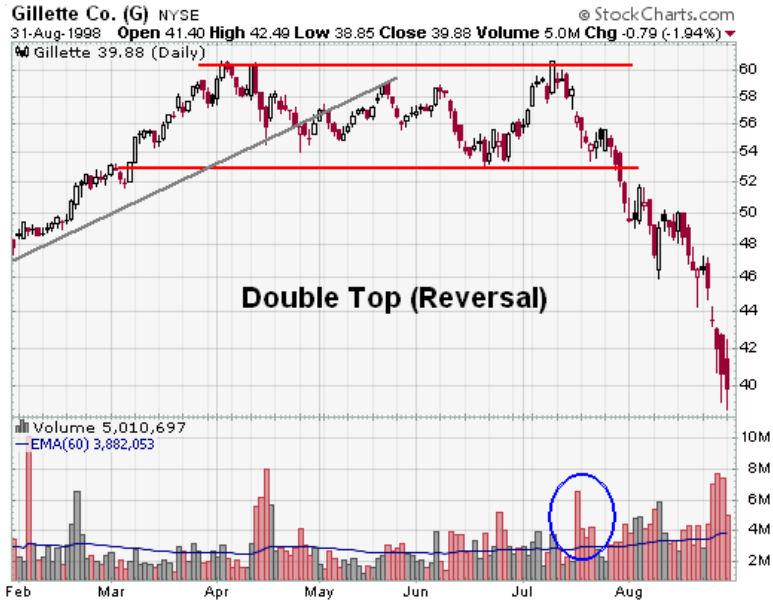

Figure 3.8: Example of the double top reversal pattern (source: *StockCharts.com*[20])

troughs instead of two peaks. An example of a double bottom pattern is given in fig. 3.9.

3.4 The Relative Strength Index

The RSI is a transformation of the time series which is used to analyse the balance of supply and demand of the underlying security. It was first invented by J. Welles Wilder in 1978.[30]

In order to calculate the RSI we first need to calculate the relative strength parameter (RS). Given the daily changes of the price in the pre-

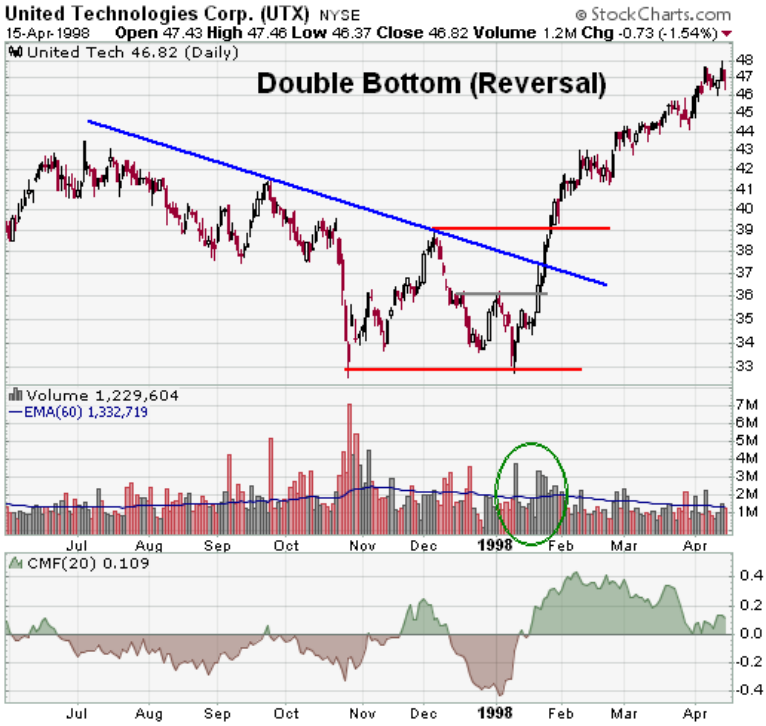

Figure 3.9: Example of the double bottom reversal pattern (source: *StockCharts.com*[20])

vious N days, the RS is computed as follows:

$$RS = \frac{\text{Average value of positive price changes}}{\text{Average absolute value of negative price changes}}$$

Subsequently, we can compute the RSI with the following formula:

$$RSI = 100 - \frac{100}{1 - 1/RS}$$

Looking at the formula, we see that the RSI oscillates within the range between 0 and 100. As for the number of days N, Wilder himself rec-

ommended a period of 14 days. Although other intervals are possible, it can generally be assumed that when someone refers to the RSI (without explicitly stating otherwise) he is implicitly refering to the 14-day RSI.

Figure 3.10: Simple RSI analysis (source: *StockCharts.com*[20])

3.4.1 Analysing The RSI

Wilders stated that in a normal market, the RSI should be expected to fluctuate in the range between 30 and 70. A value higher than 70, according to Wilder, indicates that a period of high buying pressure has taken place, and that the stock may be temporarily overbought. This could mean a negative pressure on the price, as the ratio of buy and sell activity could be expected to return to its normal range. Oppositely, a value lower than 30 might mean that the stock has had a period of heavy selling pressure,

and that the future price would tend upwards as this pressure dissipates. This type of analysis is illustrated in fig. 3.10, where we see that RSI values outside these limits coincide nicely with resistance levels in the stock price.

Wilders added a few qualifying statements to this range analysis. If the security was seen to be in a strong long-term overall uptrend, the limits should be shifted to 40 and 80. And if the stock was in a downtrend, the limits should be changed to 20 and 60.

Divergences

Wilders acknowledged that the straightforward range analysis just mentioned was a somewhat crude and not always reliable method of forecasting. But he also introduced another observation one can make from the RSI which he said was a lot more significant and reliable for predicting price moves, which is the occurence of divergences between the price and the RSI.

One type of divergence occurs when the price chart makes successively higher peaks while the RSI has successively lower peaks. This is called a bearish divergence, and such a disconnect indicates that there is a weakness developing in the uptrend, signifying that a trend turnaround could be about to happen.

The other type, which is bullish divergence, occurs when the price makes successively lower lows while the RSI makes higher lows. This could be taken as and indication that the stocks downtrend has encountered a significant rise in buying pressure (or weakening of selling pressure), and that the downtrend could be about to turn around.

Fig. 3.11 illustrates an occurence of both these phenomenons.

3.5 Elliot Wave Theory

Elliot Wave Theory is a branch of technical analysis which is used to forecast trend shifts in financial markets by identifying recurring periodic patterns in investor behavior. The theory was first developed by Ralph N. Elliot in 1938.[5]

Figure 3.11: Divergence in the RSI and price chart (source: *StockCharts.com*[20])

The theory is based on the observation that markets progress in trends, and that these trends display a certain degree of symmetry over time. Elliot stated that,

> Because man is subject to rhythmical procedure, calculations having to do with his activities can be projected far into the future with a justification and certainty heretofore unattainable.

In the theory, security prices are observed to go into cycles over various scopes of time. A supercycle takes place over multiple decades. Within a supercycle there are several normal cycles which take place over one or more years. Within these are primary cycles which lasts for several months.

According to the theory, this fractal-like structure of cycles within cycles goes down to cycles taking place on the level of minutes.

Each cycle is broken down into a set of waves (trends). For a bull market, Elliot observed a typical pattern of 8 waves taking place, the first 5 marking increased positive sentiment among investors, with the overall peak of the cycle taking place at the end of the 5th wave. For each of wave, Elliott described certain features in investor psychology and market behavior that will typically be prevalent at that stage.

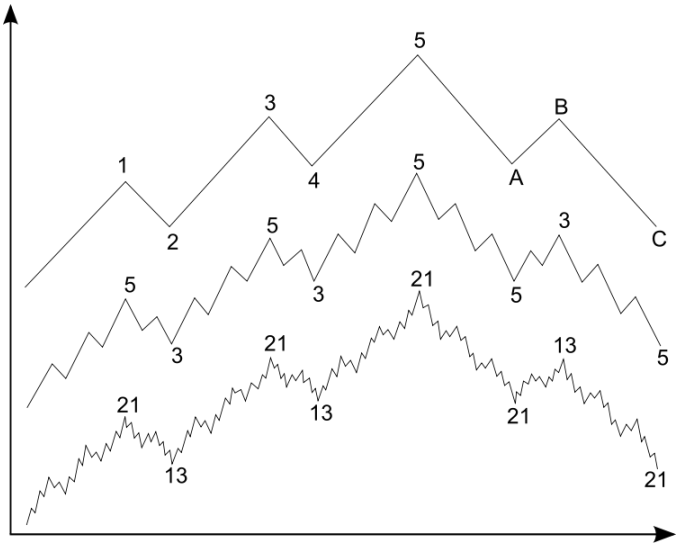

Figure 3.12: The fractal structure of Elliot wave cycles [15, p.162]

3.5.1 Fibonacci Retracements

The Fibonacci sequence, in which each number is the sum of its two predecessors $(0, 1, 1, 2, 3, 5, 8, \dots)$, plays a key role in Elliot wave analysis. When Elliot himself analysed the mathematical symmetries of waves and patterns, he was led to conclude that "the Fibonacci summation series is the basis of The Wave Principle".[5]

Looking at the Fibonacci sequence, we can note that there are certain ratios that are approximately constant along the sequence:

- The "golden ratio" of 61.8% is found by dividing one number in the series by the number that follows it (e.g. $55/89 \approx 0.6179$).
- When dividing a number by the number two places before it in the series, we get the approximate ratio of 38.2%.
- Going one step further (i.e. dividing by the element three places before) yields the approximate ratio of 23.6%.

Elliot claimed to have observed that there is a significant relationship between these ratios and the waves in his market cycle theory. Whenever the market had completed a wave, he claimed that the following (oppositely directed) wave would find support (or resistance) when the retracement was at one of these levels compared to the magnitude of the original wave. This is illustrated in fig. 3.13. Note that practitioners of this theory also count the halfway point (i.e. 50%) level as one to watch.

3.6 Volume

An important feature of the trade history that is overlooked by simply focusing on the candlestick chart itself is the volume of trading in each interval. The volume can be tricky to interpret, but it seems intuitive to think that a sudden surge in volume signifies that something significant is happening with the stock.

Some technical analysis practitioners use the volume to analyse the strength of trends. According to some literature, a trend which coincides with a surge in volume is less likely to be reversed than a trend of lighter volume.

The Percentage Volume Oscillator

The PVO is a technical indicator that is computed purely on the basis of the volume (i.e. ignoring the price). It is configured with two exponential

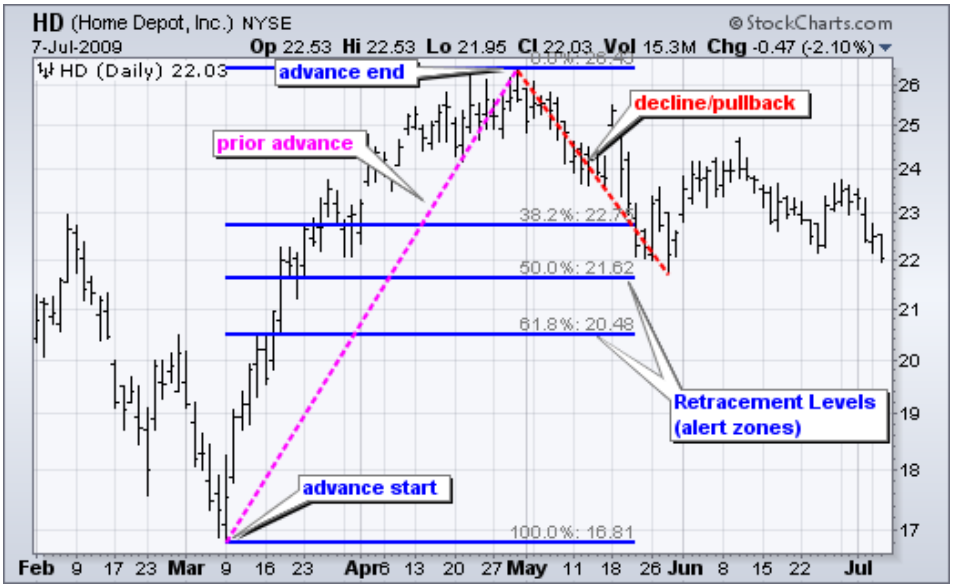

Figure 3.13: Fibonnaci retracement support levels

moving averages of the volume, of different scope (e.g. 20-day and 10-day). Given these, the PVO is computed as follows:[20]

$$PVO = 100.0 \times \frac{(\text{Longer EMA} - \text{Shorter EMA})}{\text{Longer EMA}}$$

From looking at this formula, it should be clear that the PVO oscillates about the zero axis, and that a negative PVO indicates decreasing volume, while a positive PVO indicates increasing volume.

An example of a PVO chart is given in 3.14. In this figure, the volume and PVO (using 26- and 12-day moving averages) are given for the Goldman Sachs stock in the spring of 2010. It's interesting to note the sudden spike in volume that was triggered by a relevant news event on April 16th (specifically, that was the day Goldman was sued by the US Securities and Exchange Commission on securities fraud charges).

Figure 3.14: Candlestick chart with volume and percentage volume oscillator (source: *StockCharts.com*[20])

The most common use of the PVO among practitioners seems to be for trend confirmation. If the market is in a clear overall trend, and the volume is rising overall, we can expect that trend to be less likely to be reversed. So a downtrend on heavier volume (i.e. positive PVO) is a clear negative sign, while an uptrend on heavier volume is a clear positive sign.

Chapter 4

Artificial Neural Networks

Artificial neural networks (ANNs) are a class of machine learning algorithms that draw inspiration from biological neural systems. They are generally implemented in computer software with the aim of enabling automatic learning and subsequently autonomous problem solving.

This chapter begins with brief introduction to biological nervous systems, followed by an explanation of how these systems are emulated in ABBs. It is not the aim of these explanations to get too bogged down in technical details, but rather to give readers of this report who might not be too familiar with computer science (and machine learning in particular) a chance to gain a basic intuition of the principles behind this thesis.

Having briefly explained what ANNs are, we explore some existing research in applying them to financial time series analysis. We wrap off this discussion with a brief comment on some other types of machine learning mechanisms and how they can be applied to financial forecasting.

4.1 Biological Nervous Systems

The science of nervous systems (i.e. neuroscience) is a complex body of knowledge which could fill several books (if not entire libraries), so a brief introduction like this one will necessarily be somewhat simplistic. Even

worse, there are still central questions (especially with regards to learning) which remain unanswered by scientists of the field to this day.

That being said, ANNs are by no means exact replicas of their biological counterparts. Only a very rudimentary understanding of biological neural systems should be sufficient for grasping the basic motivation behind the design of ANNs.

4.1.1 Neurons

The nervous system consists of assemblies of interconnected cells called neurons. The basic topological structure of a neuron is visualized in fig. 4.1.

Out of each neuron there grows several branch-like structures called dendrites. It is through its dendrites that a neuron is able to receive electrical signals from other neurons. There is also a single larger filament called the axon which carries outgoing signals from the cell. The axon branches out to come in contact with up to several other neurons.[6, p.167]

Where the outgoing axon meets the dendrite of another neuron, a connection is made in the form of an electrochemical device called a synapse. Electrical signals mediated through the axon triggers the release of certain chemicals across the synapse, known as neurotransmitters. When the dendrite reacts with these chemicals, a complex reaction begins which utimately results in the buildup of ionic charge in the dendrite membrane. This charge further translates to a voltage difference that is propagated through the dendrite down into the recieving neurons cell body.

Each neuron has a certain activation threshold that the sum of the incoming voltages (from the dendrites) need to exceed in order to activate an outgoing electrical signal in the axon. When this occurs, we say the the neuron is excited, or that it is firing.

There are several more complicating factors in how neurons operate that will not be discussed further here. The very simplistic picture laid out so far should be sufficient for the context of this thesis. For a more detailed and complete analysis of how neurons work, see [6].

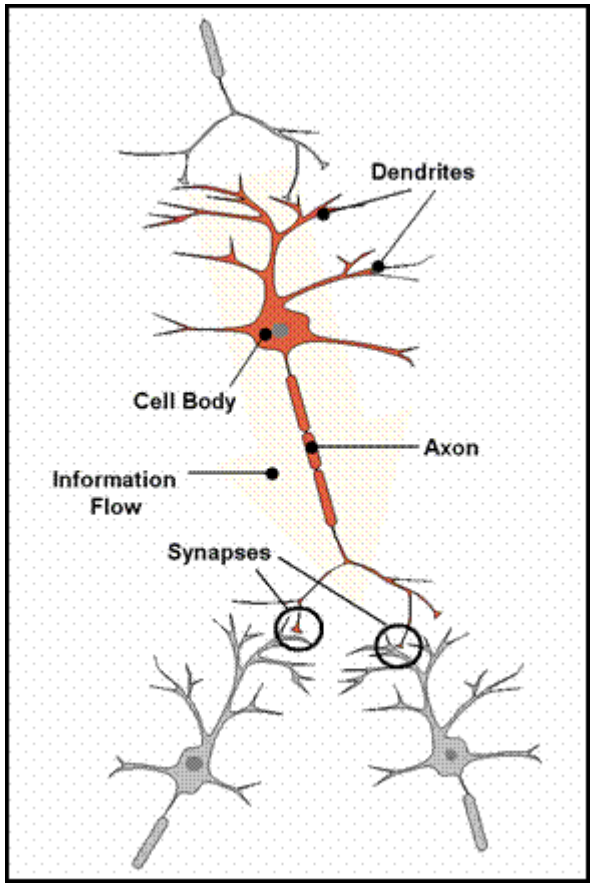

Figure 4.1: The basic structure of neurons

4.1.2 Adaptation

Neural systems give organisms the ability to adapt to their environment during their lifetime. On a macroscopic level this adaptation has several manifestations. These include things like habituation, forming associations and memorization of people and places. On a microscopic scale, what allows for adaptation are mainly the processes that affect the strengths of synaptic connections between neurons. In other words, given a neuron with several incoming connections, what will change with adaptation is which of the connections it "listens more closely" to and which of them it ignores. The exact nature of this mechanism was hypothesized more than 50 years ago by Canadian psychologist Donald Hebb:

> When an axon of cell A is near enough to excite cell B or repeatedly or consistently takes part in firing it, some growth or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased.[8, p.62]

In other words, when a source neuron is often involved in the activation of a receiving neuron, the receiving neuron will become more sensitive to signals from that source neuron. This mechanism, commonly referred to as Hebb's rule, was later verified in several experimental studies.[10] As it turns out, the opposite effect has also been shown to hold; when two neurons are connected and the source is rarely involved in the activation of the destination, the connection weakens over time as the destination neuron starts "ignoring" that source.

There are also other neurophysiological processes that can contribute to neural adaptation, such as dynamic growth and death of connections, but these normally act over much longer time periods and are much less frequent in occurence. Most computational approximations of neural systems therefore focus exclusively on Hebb-style adaptation.

4.2 Artificial Neural Networks

As was already mentioned, ANNs are computational models implemented in computer systems in an attempt to replicate some of the behavioral and adaptive features of biological neural systems.

4.2.1 Artificial Neurons

Every ANN consists of a set of units (or neurons) and a set of connections between them. Each neuron is basically just a mathematical function Φ (the activation function) that takes as parameter the activation a, which is a weighted sum of all the incoming signals to the neuron. The value of $\Phi(a)$ is the outgoing signal of the neuron.

It is important to note that the activation parameter of a given neuron is a *weighted* sum of all its incoming signals:

$$a = \sum w_i \times x_i \tag{4.1}$$

Here w_i is the weight of the incoming connection i, and x_i is the signal value that was sent by the neuron on the other side of that connection. It's clear that the higher the weight of the connection is, the more influence it will have on the neuron. In correspondence with the Hebbian principles we discussed above, it therefore seems intuitive that we can simulate adaptation by adjusting the values of these weights. This is typically done using a method called *backpropagation*, which we will come back to later.

The exact nature of the activation function $\Phi(a)$ can be defined in several different ways. One very simple and somewhat common approach is to use a step function which is either 1 or 0 based on whether the activation a is greater than some constant threshold υ, i.e.:

$$\Phi(a) = \begin{cases} 1, & \text{if } a \geq \upsilon \\ 0, & \text{otherwise} \end{cases} \tag{4.2}$$

While this approach works well enough in many situations, it is clear that more information could be produced by each neuron if its activation

function was continuous instead of just binary. This is because a binary Φ means the neuron can only take on one of two states, whereas a continuous Φ can take on any number of different values. One of the most popular choices of continuous activation functions, which is also the one used by the Krang system to be discussed in chapter 5, is the symmetric sigmoid function:

$$\Phi(a) = \tanh(k \times a) \tag{4.3}$$

Here k is a scaling factor which determines how steep the curve is. The resulting value is bound to the range $\langle -1, +1 \rangle$. Fig. 4.2 shows the shape of this function with $k = 1$.

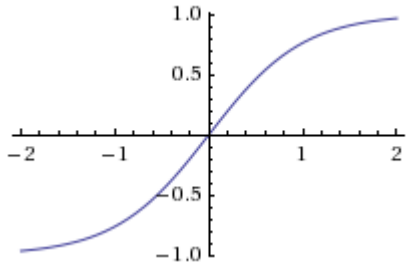

Figure 4.2: The symmetric sigmoid activation function (with $k = 1$)

4.2.2 Layer Architecture

Now that we understand the basic mechanism of how these artificial neurons operate individually, we can next consider how a network of them operates in unison. The standard way of designing ANNs is to group the neurons into N layers, including one input layer, one output layer, and up to several hidden (internal) layers. Such a network is illustrated in fig. 4.3. Notice that in this network, a given neuron in one layer is not necessarily connected to all the neurons in the next. This is what we call a *sparse* network. A *complete* network is one in which any given neuron is always connected to every neuron in the next layer.

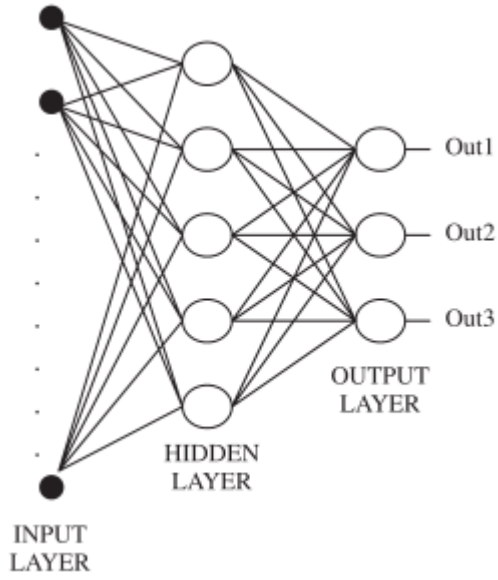

Figure 4.3: Example of an artificial neural network with layers

The Input Layer

The input layer can be thought of as the "sensor organ" of the ANN. It is where we set the parameters of the environment (i.e. the information we want the ANN to make a decision about). The neurons in this layer have no incoming connections, since their values are set from an external source. The outgoing connections send these values to the neurons of the next layer in the hierarchy.

The Hidden Layer(s)

In between the input and output layers, we put a series of one or more "hidden" layers. The reason we call them hidden is that they are invisible to any external processes that interact with the ANN. The neurons in these layers have both incoming connections from the preceding layer and

outgoing connections to the succeeding layer, and work just as described earlier in this section. The hidden layers can be thought of as the "cognitive brain" of the network.

The Output Layer

The output layer holds the end result of the computations of the ANN. If the input layer holds the parameters of a problem, the information here can be interpreted as the proposed solution. The neurons in this layer have no outgoing connections, because their Φ-values are read directly by whatever external process is using the network.

Running The Network

To test the network, we simply load the problem information into the input layer neurons, and compute Φ for every neuron in each of the succeeding layers (layer by layer untill we reach the output layer). The resulting values in the output layer will depend on what training we have previously exposed the network to.

4.2.3 Training The ANN

Our discussion of ANNs so far has explained what they do (i.e. what calculations are made) when they are given some input that is transformed into some output (as the neurons are updated through the network from the input to the output layer). In this section we will see how we can attempt to teach the network to solve specific problems by showing it examples of problems with given correct solutions.

We already mentioned briefly that biological adaptation can be simulated by varying the weights of the connections in the network. This is typically done by implementing a process called *backpropagation of error*. Qualitatively, the process can be described roughly as follows:

1. Load a an example problem from the training data.
2. Run the network normally with the problem information.

3. Calculate the error between the resulting output of the ANN and the actual correct solution.

4. Iterate backwards through the layers of the network, and slightly tweak the weights of all the connections in the direction (positive or negative) that minimizes the error of the output.

This process is repeated over a set of training data. The "tweaking" of the weights is done using a formula called the *Delta rule*, which is based on the principle of gradient descent. We won't go into further detail about the actual mathematics involved here; the curious reader can look to e.g. [6, p.221] for a more complete descritpion.

Now that we have seen what ANNs are and have a basic understanding of how they work, we can begin investigating how they can be applied to financial security analysis.

4.3 ANNs Applied To Financial Analysis

Our discussion of ANN applications to security analysis will begin with a review of some existing literature. This will be followed by a brief commentary where some potential improvements to these approaches are discussed.

4.3.1 Forex Forecasting With ANNs (Huang et. al.)

Huang et. al. published a neat review of several research attempts for applying ANNs to forecast the foreign exchange markets.[9]

Their analysis defines a general design framework consisting of three main steps, that they use as a basis of comparing the various efforts made by different researchers. These three steps are: *selecting inputs*, *preparing data* and finally *deciding ANN architecture*. Let's take a quick glance at what each of these steps should entail (according to Huang) and what choices have been explored by the research cited in their review.

Selecting inputs

According to Huang et.al., the first step in designing a neural network for forecasting a financial time series should be to decide what variables the network should take into account.

Most of the research they cite use a design referred to as a *univariate* model, where the only variable taken into account by the ANN is the price time series. A simple (and common) example to realize this is to have a time series of daily intervals, and an ANN with an input layer of N neurons and output layer with only one neuron. The input layer then holds the time series data points of N consecutive days, and after the ANN is computed the output layer would hold the prediction for the "$(N+1)$th" day.

Other researchers have attempted to implement *multivariate* models in which several other variables than just the price is take into account by the ANN. In the context of foreign exchange forecasting, examples of such variables might be forward lending rates, central bank lending rates, or the spot prices of commodities like oil or gold.

Walczak and Cerpa [23] suggested a method for isolating useful variables for multivariate networks. In their method, one first gathers as many potentially useful variables as possible, then carefully eliminates the irrelevant ones one by one by comparing network performance with and without each variable.

Smith [19] claimed that having input variables which are heavily correlated to the time series which is to be forecasted should be avoided. His reasoning is that the information such variables provide is already inherent in the time series itself, and will therefore be taken into account more than once by the ANN, leading to a biased forecast decision.

As for whether univariate or multivariate models are to be preferred, Huang et.al. don't draw any general conclusions in their review. They note that univariate models are simpler to train and need less input, but multivariate models might be more credible as long-term forecasting tools since they take a more complete picture of the financial environment into account.

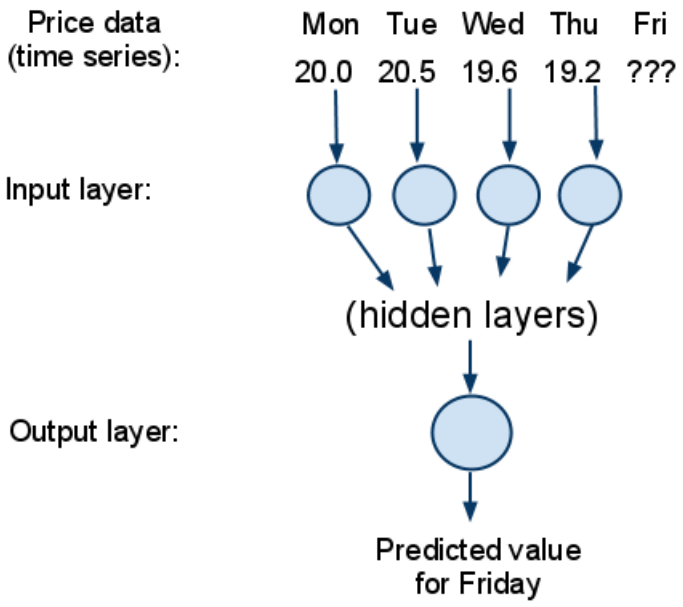

Figure 4.4: Example of simple univariate ANN

Preparing Data

After we have decided whether we are using a univariate or multivariate model (and which variables should be included for the latter), the next step is to decide how this information should be presented to the input layer of the ANN, and what a good set of training data would be.

The main question that all models (whether uni- or multivariate) need to have decided is how to present the price data to the ANN. The most common method seems to be one where the neurons of the input layer are treated as a rolling window over the time series. In a daily series, each neuron then holds the price data for a single day. A simple univariate network which uses this type of input is illustrated in fig. 4.4. In multivariate

models, the input layer may be split into several such rolling windows (one for each variable).

A central design issue for this type of model is deciding how many neurons the input vector should hold (i.e. how far into the past the ANN should look to best forecast the future). Feeding the network too many past periods will introduce too much complexity and make it harder to train the network properly, while too few periods may not be sufficient to unveil useful features in the data. According to Huang, the best method for deciding the input size is therefore a process of trial and error, where one attempts to add or subtract input neurons until simulations show that the minimum of essential neurons is found.

To compensate for the fact that the scale of the price level might vary wildly over time, which might confuse the ANN, the prices are typically normalized to relative rather than absolute values. This process is especially important for multivariate models, because here some values are likely to be on a higher level than others in absolute terms, and these will then automatically be given preference in the ANN. The most common method for normalization (at least in the research cited by Huang et.al.'s review) is to apply a so called "n-day lag" transformation to the price series, where the value corresponding to a given day is the price of that day minus the price n days prior.

Another approach entirely, which is not discussed by Huang et.al., is to analyse and rate the data heuristically using techniques from technical analysis, and to feed these "TA-ratings" to the neural net instead of the raw time series itself. This approach has been previously employed by Larsen[12], and will be discussed at greater length later in this chapter.

The last question we must ask ourselves in this phase is what sample data we should use, and how it should be split between training and simulation data. Walczak [22] did a relatively thorough investigation where he compared network performance with various sizes of training data sets. In his discussion, he claimed to have discovered what he referred to as the *Time Series Recency Effect*, which states that:

> More recent data is better suited for training (i.e. gives better

forecasting performance) than older data.

A possible explanation for this is that the predictive power of the features that are learned from observing a slice of the time series declines over time, because the underlying market forces that drive the movement of the price undergo fundamental changes over time.

Walczaks empirical results point to 2 years as being the optimal amount of training data. It may be worth noting that he used a univariate model with daily time series data in his tests.

Deciding ANN Architecture

The last step of Huang et.al.'s design framework is deciding the ANN architecture. In addition to standard feedforward networks (which is what we described earlier), they also mention two other types of ANNs as viable options. These are *feedback networks* and *competitive networks*. We will not describe the technical details of these types of ANNs further here; suffice it to say that Huang et.al. conclude their discussion with a recommendation of feedforward networks over the alternatives.

As for choosing the number and size of the hidden layers in the ANN, few guidelines are provided by Huang et.al. They simply say that trial and error is needed to determine the optimal structure for a given context. Of course, one should not carry out this trial and error process on the same data that is to be used when determining the performance of the network (as that would be "cheating"). They therefore recommend splitting the simulation data not only into training and test sets, but also a separate validation set. One would then use the training and validation data when trying to tweak performance, and go on to use the test data exclusively for the final performance evaluation.

Performance Review

Out of the 11 studies considered by Huang et.al., 4 reported mixed and inconclusive results while the remaining 7 reported statistically significant positive results. However, it is difficult to draw general conclusions on

the basis of these papers because different researchers have used different performance measures and no standardized benchmark exists by which the empirical results of these studies can be compared with any rigour.

Some of the studies also compared performance with other methods of machine learning. The details of these systems go beyond the scope of this report, so we can just note that none of them showed significantly better results than ANNs.

4.3.2 Larsen: Automatic Stock Trading Based On Technical Analysis

Larsen[12] implemented a neural network forecasting model using a novel approach which was not considered in the review by Huang et.al. His approach is based on a so-called agent architecture.

In this system, the raw time series is analysed using heuristically defined methods that are based on methods from technical analysis (as discussed in chapter 3). Each technical analysis method (e.g. moving averages) is implemented as a separate agent, which makes the necessary computations and produces an enumerated rating indicating either a buy recommendation, a sell recommendation, or a neutral rating of the stock being analysed.

In addition to this set of technical analysis agents, he defined a set of *aggregate agents* which used various machine learning methods in an attempt to learn the most efficient utilisation of the technical analysis signals taken in combination. The learning algorithms he used include a simple voting classifier, a voting classifier trained with a genetic algorithm, two different decision trees, and finally an ANN classifier.

Each of the aggregate agents were trained and tested individually, on the same training and simulation data, before the results where compiled. His measure of performance was a simulated portfolio which bought and sold the stock whenever a buy or sell rating was produced. The best performing aggregate agent turned out to be one of the voting classifiers, while the ANN-based agent had mixed results in the various tests.

The Krang system, which is described in chapter 5, uses a similar architecture to the TA-based agent design introduced by Larsen. But while

the two implementations share a lot in common (particularly Larsens ANN aggregate agent), there are still considerable differences between the two. Some of these differences arise from several observed weaknesses in Larsens system which the Krang system tries to avoid. Some of the main problems witnessed are:

- Larsen exclusively uses *daily* candlestick data. This makes it hard for his system to make observations in different time scales (both longer-term views and intraday). It would certainly be interesting to see if this type of system could be used for intraday trading, which is what a lot of the proprietary systems (HFTs in particular) seem to specialize in.

- His agents can only produce three different signals: buy, sell or neutral. This makes it simpler for algorithms like decision trees to use the agents, but for ANNs there is no good reason not have a non-discrete rating system in which each signal carries more information (so it could at least differentiate between e.g. a "weak buy" and "strong buy" signal).

- Every prediction in his system is always for the next day, while a real life chart analyst would be more likely to produce predictions for different time horizons.

- The performance measure he uses, with a simple single stock portfolio simulation, is rather crude. There are many trading strategies one could think of in using the signals for trading the stock (e.g. using mechanisms like stop-loss, where there is a limit to how much a single trade can lose). There may even be better ways of measuring the predictive power of the generated predictions which avoid simulated trading altogether.

- While it is indeed interesting to compare the effectiveness of different machine learning algorithms, one might wonder if better performance could have been attained from any individual method if there was a more concentrated effort in that implementation. The ANN agent

in particular has been given little focus in the report, compared to some of the other methods. This can be taken to indicate that the same was true for the implementation phase of his research, and that better performance could have been achieved from the ANN if more effort had been put into e.g. finding an optimal network structure. This is also supported by the fact that he doesn't mention having used validation sets like Huang et.al.[9] recommend for selecting ANN architecture (as mentioned on page 45).

4.3.3 Using ANNs For Pattern Recognition In Financial Time Series

A big challenge in making an automated technical analysis system is finding a way to reliably identify trend reversal patterns such as the head and shoulders pattern. While the presence of such patterns are often obvious for a person looking at a chart, this type of classification is notoriously tricky to implement heuristically in a computer algorithm.

Because ANNs have been applied with great success to problems like feature recognition in photographs (e.g. for recognizing faces[17]), one might think that they could be equally efficient for time series pattern recognition (as those problems are structurally similar).

Indeed, there does exists some prior research in this area. Zapranis and Samolada[18] outlined a method for implementing an automated ANN-based recognizer of the head and shoulder pattern. Aamodt and Larsen [1] tested this approach with somewhat promising initial results. Still, the problem of finding a good set of example patterns with which the networks can be trained and tested properly remains a big challenge.

Chapter 5

The Krang System

5.1 Introduction

The Krang system is a computer program that was developed for the purpose of this report. The primary aim of Krang was to enable the creation, training and testing of ANNs that are able to predict the fluctuations of financial time series with some degree of precision and reliability.

Inspired by the approach of Larsen[12], which was described in section 4.3.2, the system uses techniques from technical analysis to guide the ANNs in their forecast computations. The way this is implemented is principally similar to his agent architecture.

Krang was developed with Microsoft Visual C++ 2008. It uses libraries that are native to the Windows operating system, so the code is not cross-platform compatible. As a digression, it bears mentioning that the name of the application was inspired by a famous comic book supervillain.

5.2 System Overview

This section is intended to give a brief overview of the various components of the Krang system and how they interact.

5.2.1 The Agents

One of the main components of the Krang system is its set of agents, each of which uses a different aspect of technical analysis to evaluate the price curve. The implementation of this analysis is defined heuristically based on recommendations from technical analysis literature. The result of the analysis is stored as a signal value, which is a decimal number between -1.0 and +1.0. A negative value is taken to mean that a bearish observation was made, and likewise a positive value means a bullish observation. If the conclusion of the analysis was neutral or undecided, the signal is set to zero.

More detailed descriptions of each agent in Krang can be found later in this chapter, in section 5.3.

5.2.2 The ANN

The other major component of Krang is its neural network module. Every time the Krang application is launched, a single ANN is instantiated, either through creation of a new network (for the purpose of training) or by loading a previously trained network from disk (for the purpose of testing).

Every ANN used by Krang is configured externally in an XML file, which specifies structure of the input layer, the hidden layer(s) and the output layer. The format of these XML files are described later in this chapter (section 5.4.2).

Input Layer

The input layer of the ANN contains a set of neurons, each corresponding to one agent. The amount of neurons, and what agent each of them should use, is configured in the ANN configuration file.

Hidden Layers

The ANN created by Krang can have an arbitrary number of hidden layers, with an arbitrary number of neurons in each of them.

Output Layer

The output layer of the ANN holds the predicted price changes of the stock, for various future times. Each output neuron is assigned one time duration, which decides how far into the future its prediction should be for (e.g. 10 minutes, 1 hour, 1 day or 1 month). So if an output node assigned to 2 hours holds the value -0.015, then we can read that as a prediction that the price will change by -1.5% over the next two hours.

Typically we will only have a single neuron in the output layer of the ANN, but Krang nevertheless supports several.

5.2.3 Training Parameters

When training the network, we first have to set two parameters for the backpropagation training algorithm. These are the *learning rate* and the *training momentum* parameters. Both of these values have to be set somewhere in the range $\langle 0, 1 \rangle$.

The learning rate describes how much each training example is allowed to affect the connection weights. A lower value means that weights are only changed slightly, while a larger value means that weights change considerably.

If the training momentum is non-zero, the change in the connection weight on one iteration will depend slightly on not only that example, but also how much it changed in the previous iteration. This way, the delta of change in the connection weights are somewhat smoothened over time.

There is no general formula for finding good values of the learning rate and momentum. The normal approach is simply a process of trial and error.

5.2.4 The Simulation Loop

When the Krang application is launched, it begins by loading its configuration parameters from an external XML file. These parameters describe what simulation data should be used (i.e. which stock and over what time period), whether the simulation is for *testing* or *training* the ANN, and what ANN to use. Once these parameters are parsed and the simulation

environment has been initialized accordingly, Krang enters into its simulation loop.

The simulation trade data is a list of every trade that took place in the specified time period for the stock (specified by time, price and volume). Starting from the beginning of this list, the simulation begins iterating and step-by-step produces a candlestick series with 1-minute intervals that we will refer to as the *atomic candlestick series*. Every time a new candlestick is added to the atomic series, all the agents re-analyse the data and update their signals. Once this is done, the ANN is updated with the new signals.

It's worth noting that in early versions of Krang, the simulation loop was designed without the concept of an atomic candlestick series, and so everything (the agents and the ANN) was updated on every trade (instead of on every minute passed). It quickly became apparent that this was not a good solution, for several reasons. One problem was that some of the stocks have such a high number of trades (over a million per year in some cases) that updating all the agents and the ANN with each trade simply took too much time. Also, when keeping record of the generated ANN predictions, it is much easier to generate meaningful statistics from them if they are sampled at uniform time intervals (i.e. 1-minute) than sporadically (whenever trades might occur). Both of these issues are resolved by having the atomic candlestick series.

The process flow of a Krang simulation is summarized in figure 5.1.

Initialization:

→ Load the trades for the specified stock and time period.
  Training only:
  → Create a new ANN with the specified architecture.
  Testing only:
  → Load a previously trained ANN.

→ Instantiate all the agents that are required by the ANN.

Loop:

→ Generate the next atomic candlestick (using the trades that occur in the time interval it represents).

→ Update all the agents.

→ For every output node in the ANN, peek ahead in the trade data to see what the correct prediction would be.

→ Load the agent signals into the input layer.
  Training only:
  → Set the output layer nodes to their correct predictions.
  → Run the training algorithm (i.e. backpropagation).
  Testing only:
  → Run the network so all the neurons get updated.
  → Log the computed forecasts from the output layer, and their error.

→ If there are no more trades, the simulation ends. Otherwise, run the loop again.

Figure 5.1: Process flow description for Krang simulations

5.3 The Agents

This section provides a detailed description of all the agents that were implemented in the Krang system. As mentioned earlier, each agent uses a different method of technical analysis to generate a signal value from -1 to +1, indicating whether it perceives a positive or negative bias on the stock price.

5.3.1 The `SPSupportAgent`

The `SPSupportAgent` uses the peaks and troughs (swing points) of the price curve to look for support/resistance levels, in a way similar to that described in section 3.1.

To identify the swing points on a curve, an algorithm was implemented which analyses the back of the curve and stores a swing point any time a swing has been made greater than a given threshold (e.g. 15%) in the opposite direction of the previous peak/trough. This algorithm was implemented in the separate support class `SwingPointVector`, which is used by this and several other agents to identfy the swing points of the curve.

On every update, the agent looks at the past few swing points and tests to see whether the price curve is currently approaching one of these levels. If one of these levels are being approached from below, the agent generates a negative signal (expecting resistance). If a level is being approached from above, the agent generates a positive signal (expecting support).

5.3.2 The `TrendLineAgent`

The `TrendLineAgent` uses the swing points of the curve to try and identify potential support or resistance trendlines, as described in section 3.1.

The heuristic of this agent calculates the line extended from the last two peaks and last two troughs and tests too see whether the price curve is approaching either of these lines. If the agent is approaching support, a positive signal is generated. If it's approaching resistance, a negative signal is generated.

5.3.3 The `MATrendAgent`

The `MATrendAgent` is an agent class which was implemented so as to produce signals based on the trend classification methods outlined in section 3.2.3.

To calculate the moving average, the support class `MovingAverageTransform` is used, which computes and stores a simple moving average of the candlestick series. On every iteration of the simulation loop, the moving average is updated to reflect the latest trade information.

The `MATrendAgent` combines three such moving averages of different interval sizes (e.g. 20-day, 15-day and 10-day). It's resulting agent signal value (in the range $[-1.0, +1.0]$) depends on whether the stock is in a downtrend (negative value) or uptrend (positive value).

The classification is based on a comparison of the three moving averages. Let's denote the longest average by the variable L, the shortest average as S and the middle average as M. The trend classification heuristic makes the following comparisons to determine the signal value:

- If $L > M > S$, then the stock is in a clear downtrend, and the value is -1.0.
- If $L < M < S$, then the stock is in a clear uptrend, and the value is $+1.0$.
- If none of these holds, we check whether $L > M$, $L > S$ and $M > S$. If two of these three conditions are true, we classify an uncertain downtrend, with value -0.5. If not, we classify and uncertain uptrend with value $+0.5$.
- If the moving averages are exactly equal (extremely unlikely), the returned value is 0.0.

5.3.4 The `MASupportAgent`

The `MASupportAgent` uses a moving average to look for support or resistance levels on the curve. As discussed in 3.2.3, the general idea is that

when the candlestick series approaches the moving average, it will tend to encounter some resistance (if approaching from below) or support (if approaching for above).

The agent is configured with a moving average of a given time horizon, as well as a proximity threshold (given as a percentage) of how close the curve has to be to the moving average for the classification to work. The actual heuristic used to calculate the signal value is a little too involved to be described in full detail. One thing it does is look back a certain amount of candles to check that the moving average was not already pierced. A negative signal is then generated if it is determined that the curve is approaching from below (and we expect the moving average to provide resistance), and the signal is positive if the curve approaches from above (for support).

5.3.5 The `RSILevelAgent`

The `RSILevelAgent` is based on the RSI of the price curve, as discussed in section 3.4. This agent computes the RSI (using the `RSITransform` support class) and generates its agent signal based on whether the RSI is found to be in oversold or overbought territory.

The heuristic which generates the signal is quite simple. If the RSI is found to be greater than 70, a positive signal is produced in linear proportion to how much the RSI exceeds this threshold. A negative signal is produced in the same manner when the RSI is below 30.

5.3.6 The `RSIDivergenceAgent`

The `RSIDivergenceAgent` is also based on the RSI, but this agent uses the somewhat more sophisticated approach of looking for divergences between the RSI and the price curve (see p. 3.4.1).

The agent computes the RSI series and identifies the swing points (peaks and troughs) of the curve. It then compares the peaks and troughs of the RSI with the peaks and troughs of the candlestick series. If the price curve is found to make successively higher peaks while the RSI makes successively

lower highs, a negative signal is produced by the agent. Oppositely, a positive signal is produced if the price curve makes lower troughs while the RSI makes higher troughs.

The agent is configured with an RSI series (typically 14-day), and a swing point identification threshold.

5.3.7 The `FibonacciAgent`

The `FibonacciAgent` is based on the principle of Fibonacci retracements, which was discussed in the context of Elliot Wave Theory in section 3.5.1. To generate its signal, the agent computes the difference between the previous peak and trough of the price curve, and calculates the appropriate support/resistance levels using the Fibonacci retracement ratios (0.236, 0.382, 0.500, 0.618 and 1.0). Some of these ratios are given a higher weighting than others (e.g. the golden ratio of 0.618 is seen as more significant than 0.236). Whenever the curve approaches one of these levels, a signal is generated based on whether the level acts as resistance (negative signal) or support (positive signal).

5.3.8 The `VolumeAgent`

The `VolumeAgent` analyses the PVO (section 3.6) and compares it to the current trend of the price curve (which is classified by comparing moving averages).

If the PVO is positive, a signal is produced in proportion to the PVO. If the curve is in an uptrend, the signal is positive, and oppositely negative for a downtrend. If there is no clear trend, or the PVO is negative, the signal is set to 0.

The agent is configured with two interval counts, one for each of the volume moving averages that are used by the PVO.

5.3.9 The `DoubleTopAgent` and `DoubleBottomAgent`

The `DoubleTopAgent` and `DoubleBottomAgent` look for the occurence of either double top- or double bottom trend reversal patterns, respectively.

This is done heuristically by testing for a prior trend and analysing the three latest swing points of the curve to see if they fulfill the required features of these patterns, as described in section 3.3.

During preliminary tests, it was observed that the patterns were recognized too rarely by these agents for them to be of any particular use in the ANN training process. It may be that the heuristics used were too strict in how they tested for the required properties of the patterns. Using pattern recognition ANNs (as mentioned in section 4.3.3) might have been a better way to go about this problem.

5.4 Using Krang

In this section, the Krang interface is introduced. We will start by looking at the various graphical components of Krang and how they can be intepreted and manipulated to obtain information about a simulation. Next we will describe how simulations can be configured and customized.

5.4.1 The Graphical Interface

When the Krang application is launched, it automatically reads the parameters of the simulation to be performed from a configuration file and runs the simulation. During (and after) the simulation, three main windows are displayed with various controls and information. The overall view looks something like the screencap in fig. 5.2.

The Status Dialog

The top left window is the status dialog. This window contains messages from the core engine of the simulations, as well as a progress bar indicating approximately how much of the simulation is done (and how much is left). It also has button controls to pause, resume and exit the simulation.

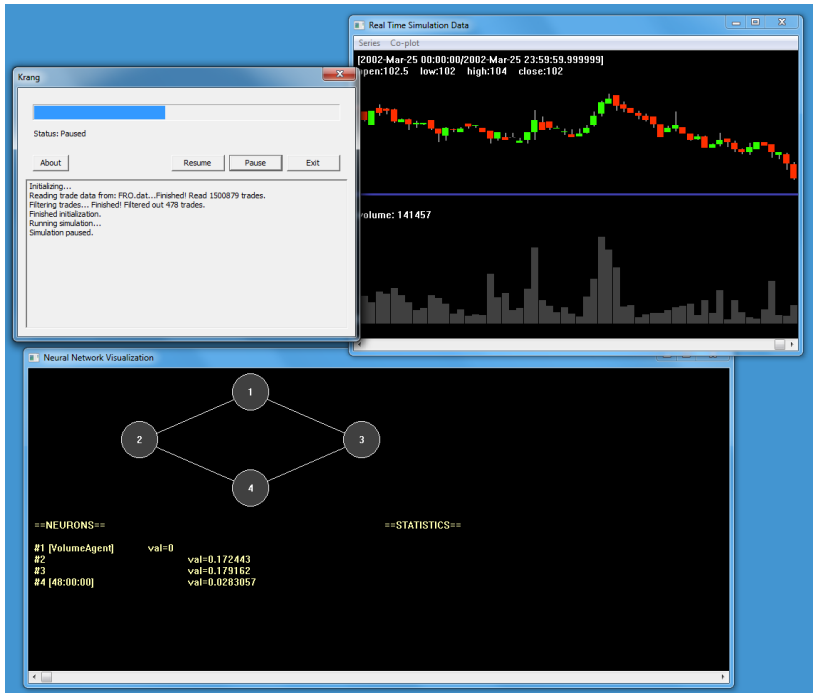

Figure 5.2: The three main windows in the Krang application

The Simulation Data Window

The top right window primarily displays the trade information processed by the simulation. Every candlestick series (e.g. hourly, daily) used in the simulation can be selected from the "Series" menu. The selected series is displayed in the main area of the window.

In the lower part of this window is the companion plot of the candlestick series. The default display is the volume, but you can also select the various technical analysis agents that are active in the simulation. This plot will then display the signal generated by the agent (vertically synchronized to the candlestick series above). In figure 5.3, the signals generated by a `VolumeAgent` are displayed in the companion plot.

Figure 5.3: The simulation data window, displaying a daily candlestick series (above) and the signals generated by a `VolumeAgent` (below)

At the bottom of the window is a scrollbar that can be used to browse the available history of the data. When you move the mouse over the window, the information text changes to describe whatever time interval you're currently hovering.

The ANN Window

The bottom of three windows in fig. 5.2 is the ANN window. This window displays the structure of the neural network, with each neuron drawn as a separate numbered circle. Below this graph, the current activation value in each neuron is listed. For neurons in the input layer, the corresponding agent class is listed next to its number. For neurons in the output layer, the time length of the neuron is displayed.

As you can see, the simple ANN in 5.2 consists of 4 neurons; 1 in the

input layer, 2 in the single hidden layer, and 1 in the output layer. The input neuron is connected to the `VolumeAgent`, and at this particular instant of the simulation, that agents signal is equal to 0. The output neuron holds the networks predicted values for 48 hours into the future.

The Simulation Statistics Log

When a simulation has finished, Krang displays the results of the sessions in a new window. When the simulation is for training an ANN, the mean squared error of the network on the training data is displayed. When the simulation is for testing a previously trained ANN, more sophisticated statistics are computed to evaluate network performance. These statistical values are described in the next chapter (section 6.3).

5.4.2 Configuration

As previously mentioned, Krang simulations are configured through external XML files. XML files are normal text files that encode data following a well-defined format[21], which allows the data to be interpreted easily by both computers and humans.

The Simulation Configuration

When the Krang application is executed, it starts with reading the parameters of the simulation configuration file. This file contains the following information:

- The simulation type, which can be either “training” or “test”.
- What data the simulation should use. This includes the ticker symbol, and the range of dates for which data should be loaded.
- The name of the neural network. The actual network configuration is loaded from another XML file with this name (e.g. if the name is “MyANN”, then Krang will look for the file “MyANN.xml” for the ANN configuration).

```
<?xml version="1.0" encoding="utf-8"?>
<simulation type="train">
  <data ticker="DNBNOR" from="2003/01/01" to="2005/01/01"/>
  <ann name="testann"/>
  <traincfg load="true" lrate="0.40" lmom="0.05"/>
</simulation>
```

Figure 5.4: An example simulation configuration

- Configuration parameters for the training algorithm. This includes the learning rate and momentum.
- A boolean "load" parameter indicating whether Krang should try to load previously trained connection weights before training. If this is set to "false", the connection weights are initially randomized.

An example of how this file might look is given in fig. 5.4. In this example, the simulation will perform network training on a network named "testann", with data from the "DNBNOR" stock in the time period 2003-2005. Krang will attempt to load previously trained connection weights for "testann" before training.

The ANN Configuration

The configuration file for a neural network contains the following information:

- A list of agents for the input layer.
- A list of hidden layers, and their sizes.
- A list of prediction time frames for the nodes in the output layer.

An example of an ANN configuration file is given in fig. 5.5. This network contains four agents: two `MATrendAgent`'s), and two `MASupportAgent`'s. Further, there are two hidden layers, with sizes 4 and 3. The output layer contains two nodes: one for making 2-hour predictions and one for making 24-hour (i.e. 1-day) predictions.

```
<?xml version="1.0" encoding="utf-8"?>
<ann name="testann">
  <input>
    <agent class="MATrendAgent" duration="01:00:00" shortint="10" longint="30"/>
    <agent class="MATrendAgent" duration="24:00:00" shortint="5" longint="15"/>
    <agent class="MASupportAgent" duration="01:00:00" intervals="20" threshold="0.0025"/>
    <agent class="MASupportAgent" duration="01:00:00" intervals="10" threshold="0.0025"/>
  </input>
  <hidden>
    <hlayer size="4"/>
    <hlayer size="3"/>
  </hidden>
  <output>
    <out duration="02:00:00"/>
    <out duration="24:00:00"/>
  </output>
</ann>
```

Figure 5.5: An example ANN configuration

5.4.3 Simulation Data

The stock price data used by Krang in its simulations is loaded from an external file based on the ticker symbol and date range given in the simulation configuration. The data files contain a binary array of trades, each trade an instance of the following C-structure:

```
struct Trade {
   time_t time ;   // 64-bit POSIX time
   double price ;
   unsigned long volume ;
} ;
```

For anyone reading this report with the intent of utilizing the Krang system with custom data, just make sure it's prepared as an array of such structures and Krang should be able to make use of it. The filename should be the stock ticker and the file extension should be ".DAT" (e.g. the trade data of the stock DNBNOR should be stored in the file "DNBNOR.DAT").

Data Filtering

When Krang has finished reading the trade data, the first thing it does is to filter it for irregular trade records (so-called "blips"), that might have

occured because someone entered an incorrect order to the exchange (or due to some other error). This filtering is performed by iterating the trade data from beginning to end, and for every trade with a price which deviates more than some threshold (say, 20%) from the trades surrounding it (before and after), that trade is removed.

Chapter 6

Simulations

The purpose of this chapter is to describe how the Krang simulations were set up in order to generate the results which are presented in the following chapter. We begin by discussing the data that was used to train and test the ANNs. Next, we describe how the ANNs were designed and trained. Lastly, we discuss what tests will be performed, and introduce the various performance measures that are used to evaluate the ANNs.

6.1 Simulation Data

The data used for the simulations described in this report consists of records of every trade that occured in 10 separate stocks traded on the Oslo Stock Exchange (OSE) from January 1st 1999 to January 1st 2009. The data was procured from the OSE upon request. They agreed to release the data for a small fee (to cover the work expenses they incurred when extracting and transferring the data). We also had to sign a contract stating that we would not release the data publicly. The stocks for which data was procured are summarized in table 6.1.

The data received was separated into several text files, one for each year. Each row in the text file contained information about one trade (the ticker symbol of the stock, the exact date and time the trade took place,

Company	Ticker	Trades 1999-2009
DnB NOR	DNBNOR	1,787,879
Frontline	FRO	1,500,879
Golden Ocean	GOGL	1,087,206
Jinhui	JIN	524,701
Norwegian	NAS	81,220
Norsk Hydro	NHY	2,965,536
Orkla	ORK	1,776,054
Storebrand	STB	1,017,696
Yara	YAR	2,079,704

Table 6.1: Overview of the 10 stocks which are used in the simulations.

the price of the trade and the number of stocks that was traded). The data was translated to the binary format described in section 5.4.3 so that it could be utilized by the Krang system.

6.2 Test Plan

While the Krang system has the capability to use the same ANN for several simultaneous predictions over different time horizons, it was decided that the empirical evaluations should focus on ANNs that only make a single prediction (i.e. output layer node). This is because finding an optimal network for a single output node is easier than for several, and this approach therefore seems the most likely to succeed.

The empirical testing will be performed on 4 different ANNs, each with a different configuration of agents for the input layer, different hidden layer structures, and most importantly different time frames for price predictions in the output layer node. The different prediction time frames that will be tested are: 1 week, 2 days, 2 hours, and 30 minutes.

6.2.1 Network Design Phase

For each network, a preliminary design phase will be carried out in which the input- and hidden layer configurations are tweaked to give the best possible network performance on the stock data from January 1st 1999 to January 1st 2003. The data for this period has been reserved specifically for this purpose, so that we can test the resulting network arhcitectures on the rest of the data, for which they haven't been specifically tweaked. This allows us to evaluate the performance of the networks with good scientific rigour.

The most optimal ANNs configurations that are obtained from this process are stored so they can be used in the empirical testing phase. Note that the trained connection weights of the networks are reset to random values after this process, so it's only the configuration of the input layer agents and number of nodes in the hidden layer(s) that are stored at this point.

6.2.2 Empirical Test Phase

In order to evaluate the performance of each ANN, training and testing on the data from 2003-2009 will be performed as follows:

- Initially, the connection weights in the ANN are randomized to small positive values in the range $\langle 0.005, 0.020 \rangle$.
- The network is then trained on the data from $01/01/2003 \rightarrow 01/01/2005$, and this is repeated until no improvement is seen in the mean squared error over the same data.
- The network is subsequently tested on the data from 2005, and the evaluation results are recorded.
- The network is then retrained on the data from $01/01/2004 \rightarrow 01/01/2006$, and tested on the data from 2006.
- This process of retraining and testing is repeated in one year increments until the network has been tested on the 2009 data.

This process is carried out twice for each network, using data from two of the available stocks (picked arbitrarily).

The reason why training and testing is to be carried out in a cyclical fashion of one year increments, and not just in a single iteration, is to compensate for Walczak's Time Series Recency Effect (see section 4.3.1).

6.3 Performance Evaluation

In order to assess the predictive performance of the predictions generated by the ANNs, we need to have some accumulative measure of how well the individual predicitions tend to match the actual development of the stock price. There are two main types of evaluations that can be used for this purpose. One is to use the ANN price predictions to somehow simulate trading in the stock and then record the profit or loss that is obtained. The other way is to record the predictions and compare them to the actual evolution of the price curve using statistical analysis.

6.3.1 Simulated Trading

Larsen[12], who had a conceptually similar system to Krang, used the binary buy/sell signals generated by his system to simulate a trading position in which he bought and sold the stock continuously according to the signals. To evaluate performance, he then recorded the accumulated yield that was achieved from the beginning of the simulation to the end.

A problem with this approach is that there are nearly countless concievable strategies that could be used to trade based on the signals. For example, should we allow for short positions? Should the available capital always be invested fully in the stock, or should the size of the position change gradually? Should we allow for leverage (i.e. borrowing money so we get position sizes over 100%)? Should there be some type of stop-loss mechanism with each trade?

Another issue that should be considered is that poor predictions early on in such a simulation tend to adversely affect the yield, even if the performance was better for the majority of the simulation. To take an extreme

example; if the position loses 60% in the first month of the simulation, but then rises 100% the following year, it will still be counted as a poor performance (20% loss) even though the individual predictions may have been correct for the majority of the time.

6.3.2 Statistical Analysis

Most other researchers that have tried to apply ANNs for financial time series forecasting seem to have opted for the second type of performance evaluation; statistical analysis. This is evidenced by Huang et.al.'s review[9], in which all the cited research used at least one statistical measure, with only one out of the 11 papers reviewed having performed any form of trading simulation (it also used statistics).

Because of the problems already mentioned with simulated trading, and the fact that statistical analysis seems like standard way to evaluate performance, it was decided that the Krang simulations should also use statistical analysis as its primary way to measure performance.

During a simulation, Krang records the predictions of the ANN at 1-minute intervals along the financial time series. With every prediction, it also records what the correct prediction would be at that point by peeking ahead in the simulation data. When the simulation is over, it then has two sets of data points that form the basis of the analysis: the actual time series values X and the predicted values Y. Three separate statistical coefficients are calculated based on this data: the average error, the hit rate, and the Pearson correlation.

Average Error

The average error is calculated by taking the average of the absolute error in each prediction $y_i \in Y$. Given that the number of recorded predictions is $|Y| = |X| = N$, this value is calculated using the formula:

$$\overline{err} = \frac{1}{N} \sum_{i=1}^{N} |y_i - x_i| \tag{6.1}$$

The average error is interesting because it gives a very direct measure of what scale of error we should expect for any given prediction. We could've instead used the (more common) standard deviation, which is based on a similar principle, but the average error is somewhat easier to interpret directly.

The Hit Rate

The hit rate measures the tendency for a prediction y_i and the corresponding time series value x_i to have the same sign (+ or -). To formulate this mathematically, we must first define the binary *hit* function as follows:

$$hit(x, y) = \begin{cases} 1 & , \ x \text{ and } y \text{ same sign} \\ 0 & , \ x \text{ and } y \text{ opposite sign} \end{cases}$$

Using this function, we can write out the formula for the hit rate as follows:

$$r = \frac{1}{N} \sum_{i=1}^{N} hit(x_i, y_i) \tag{6.2}$$

Put in other words, the hit rate is simply the fraction of the predictions that have the correct sign.

If the predictions were truly random, or there was no real correlation to the curve, we would expect the hit ratio to hover around the 50% level. So in the empirical results, we should hope to see hit rates significantly and consistently above that level in order to confirm predictive performance.

Pearson Correlation

The Pearson correlation factor is a standard statistical measure of how well two separate data series' tend to correlate; in the context of these simulations, we can think of it as the tendency for the predicted time series to move in the same direction as the actual time series (and at the same rate). This is probably the most important of the three measures because

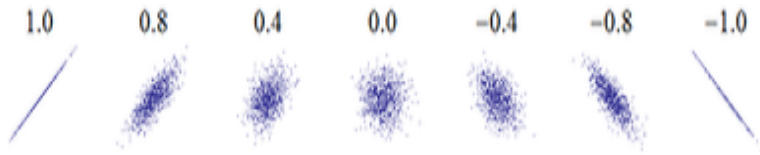

Figure 6.1: Scatter plots for two data series and the corresponding Pearson correlations

it gives a very direct and unbiased measure of whether there is a definite relationship between the predicted and actual values or not.

The correlation factor for X and Y, $\rho_{X,Y}$, is directly related to their covariance:

$$cov(X,Y) = \frac{1}{N}\sum_{i=1}^{N}(\mu_X - x)(\mu_Y - y)$$

$$\rho_{X,Y} = \frac{cov(X,Y)}{\sigma_X \sigma_Y} \tag{6.3}$$

(Using standard statistical notation, μ here denotes expected value and σ is standard deviation.)

The Pearson correlation coefficient will always lie in the interval $\langle -1, +1 \rangle$. Figure 6.1 shows how the various ranges can be interpreted. In general, a highly negative value means there is some inverse relationship between the two series (i.e. they move in separate directions). A highly positive value means that there is a linear relationship. A value close to zero means that there is no relationship, and that the two series are uncorrelated. As such, we should see values of ρ signficantly higher than zero from the Krang simulations if our ANNs work as we would hope.

Filtered Statistics

It would be interesting to see if the predictions of the ANNs are more reliable when they are significantly higher or lower than average. In other

Figure 6.2: Example of predictions from a 2-hour ANN

words, we would like to find out if there is more reason to put trust in a prediction of, say, 4% rise than a prediction of 1% rise.

To do this, we generate a separate set of predictions $Y' \subset Y$ by first calculating the average absolute value of all predictions $y \in Y$ and only picking out those predictions with greater absolute value than the average into Y'. The actual time series values for the predictions in Y' are selected into $X' \subset X$.

We then substitute $Y \to Y'$ and $X \to X'$ and recalculate the average error $\overline{err}$, the hit rate r and the Pearson correlation coefficient ρ as before.

If the hypothesis is correct, and there is a genuinely higher predictive performance for higher absolute predictions, we should expect to see this reflected in these variables (especially the hit rate and correlation). We might still expect a higher average absolute error though, simply because the predictions are known to be higher on average.

Example Calculation

To further illustrate how the statistics generated by Krang work, let us consider a simple example of how they are calculated.

Consider an ANN trained to make 2-hour predictions, being tested on the intraday chart in figure 6.2. The Krang simulation samples new fore-

Time	Predicted 2-hour change	Actual 2-hour change
10:00	$131.45 \rightarrow 131.65(+0.15\%)$	$131.45 \rightarrow 130.80(-0.49\%)$
12:00	$130.80 \rightarrow 130.20(-0.46\%)$	$130.80 \rightarrow 131.05(+0.19\%)$
14:00	$131.05 \rightarrow 131.25(+0.19\%)$	$131.05 \rightarrow 131.00(-0.04\%)$
16:00	$131.00 \rightarrow 130.50(-0.48\%)$	$131.00 \rightarrow 129.20(-1.37\%)$

Table 6.2: Sampled predictions from the example ANN

casts every minute, so there would normally be over 500 separate predictions recorded just for this day. To keep the calculations manageable, we will only sample four predictions in our example.

The sampled predictions are highlighted in red. We see that at 10:00 this ANN predicts the price to go slightly up in the next two hours. At 12:00, the ANN predicts the price to go slightly down in the next two hours, and so on. The four sampled predictictions and the corresponding price changes that actually occured, are listed in table 6.2.

We see that in this example, the ANN only had one prediction with the same sign as the actual change (sampled at 16:00). The hit rate r is therefore:

$$\begin{aligned} r =& \frac{1}{4}\sum_{i=1}^{4} hit(y_i, x_i) \\ =& 0.25 \times [hit(0.15\%, -0.49\%) + hit(-0.46\%, 0.19\%) \\ & + hit(0.19\%, -0.04\%) + hit(-0.48\%, -1.37\%)] \\ =& 0.25 \times (0 + 0 + 0 + 1) = 25\% \end{aligned}$$

It is also trivial to compute $\overline{err}$ (the average error) for this example. Simply taking the average absolute difference between the predicted and

actual changes, we get:

$$\begin{aligned}\overline{err} =& \frac{1}{4}\sum_{i=1}^{4}|y_i - x_i| \\ =& 0.25 \times \Big(|0.15\% - (-0.49\%)| + |(-0.46\%) - 0.19\%| \\ &+ |0.19\% - (-0.04\%)| + |(-0.48\%) - (-1.37\%)|\Big) \\ =& 0.25 \times (0.64\% + 0.65\% + 0.23\% + 1.85\%) = 0.84\%\end{aligned}$$

This means that the predictions were wrong by 0.84% on average.

Computing the Pearson correlation is a bit more involved (especially since we first need to find the mean and the standard deviation of both the predictions and actual changes). Writing out all the steps in these calculations seems a little tedious, but based on the two previous examples you have hopefully understood how the sample predictions and actual changes are put into the formulas.

Putting the predicted and actual changes of table 6.2 into formula 6.3, we compute the covariance to be ~ 0.058 and subsequently the Pearson correlation around ~ 0.2 (these calculations were performed in Microsoft Excel).

The Pearson correlation is actually of somewhat limited analytic value when working with such a low amount of data points. As was already mentioned, this example has only four sampled values, while a Krang simulation running over a year will generally sample well over 100,000 separate predictions.

Chapter 7

Results

Each of the four ANNs were first designed individually. This design process was described in section 6.2.1. The resulting network architectures are described in some detail for each of the networks below.

After the design phase, the empirical simulations were carried out according to the test plan (section 6.2.2). For each test simulation, filtered and unfiltered values of the average error ($\overline{err}$), the hit rate (r) and the Pearson correlation factor (ρ) where computed and recorded. They are presented for each of the ANNs below.

7.1 The 30-Minute Prediction ANN

7.1.1 Network Architecture

Finding a good network structure for the 30-minute ANN proved very tricky. After a lot of experimentation, it was concluded that the number of inputs should be kept few and simple. Factors like RSI seem to have little consequence on this short term scale.

There where four agents that were used for the input layer. Three of these where `MATrendAgent`'s, using different time frames for classifying the current trend. The last agent was an `SPSupportAgent`, with a swing point threshold of 8%. The single hidden layer of the ANN had only three nodes.

	Unfiltered			Filtered		
Year	ρ	$\overline{err}$	r	ρ	$\overline{err}$	r
2005	-0.009173	0.003529	0.5841	-0.04007	0.004654	0.5795
2006*						
2007	-0.02801	0.003927	0.478	-0.03144	0.004194	0.4829
2008	-0.01809	0.01077	0.4789	-0.0162	0.01081	0.4601

Table 7.1: Results for the 30-minute ANN with NHY stock data

	Unfiltered			Filtered		
Year	ρ	$\overline{err}$	r	ρ	$\overline{err}$	r
2005	0.1138	0.003222	0.5873	0.1557	0.003576	0.6589
2006	0.02204	0.01044	0.3325	0.02056	0.01399	0.3282
2007	0.0505	0.003229	0.4584	-0.0325	0.003719	0.4412
2008	-0.01688	0.008003	0.4949	0.02262	0.009915	0.4775

Table 7.2: Results for the 30-minute ANN with DNBNOR stock data

The learning rate was configured relatively high compared to the training momentum when performing training on this ANN. The values used were 0.6 for the learning rate and 0.1 for the momentum.

7.1.2 Empirical Results

The simulations of the 30-minute ANN were carried out with the stocks NHY and DNBNOR. The results are listed in tables 7.1 and 7.2, respectively. Note that no data was generated for NHY in 2006, since the company underwent structural changes leading to a sudden drop in 90% of the stock price, making the data useless for testing.

	Unfiltered			Filtered		
Year	ρ	$\overline{err}$	r	ρ	$\overline{err}$	r
2005	0.0845	0.003521	0.5243	0.08073	0.00396	0.5238
2006	0.0434	0.00304	0.5084	0.08152	0.003702	0.5323
2007	-0.0215	0.004025	0.4878	-0.03101	0.003229	0.457
2008	0.03989	0.007747	0.4948	-0.01056	0.006032	0.5011

Table 7.3: Results for the 2-hour ANN with DNBNOR stock data

	Unfiltered			Filtered		
Year	ρ	$\overline{err}$	r	ρ	$\overline{err}$	r
2005	0.07532	0.002316	0.5452	0.0421	0.002091	0.5331
2006	0.03912	0.002044	0.5335	0.01334	0.002267	0.5543
2007	-0.0152	0.003955	0.4993	0.02893	0.004561	0.4876
2008	0.02285	0.003099	0.5359	0.01404	0.003448	0.5044

Table 7.4: Results for the 2-hour ANN with STB stock data

7.2 The 2-Hour Prediction ANN

7.2.1 Network Architecture

For the 2-hour ANN, the same network architecture was used as with the 30-minute, with the addition of one `FibonacciAgent` in the input layer. The learning rate was lowered slightly to 0.5 while the momentum was held at 0.1.

7.2.2 Empirical Results

The simulations of the 30-minute ANN were carried out with the stocks DNBNOR and STB. The results are listed in tables 7.3 and 7.4, respectively.

7.3 The 2-Day Prediction ANN

7.3.1 Network Architecture

In the 2-day ANN, a total of 12 agents were employed for the input layer. They are summarized in the following list:

- Two `MATrendAgent`'s; one for a longer term trend classification, and one for the shorter term.
- Two `MASupportAgent`'s using the 20-day and 10-day moving averages to check for support/resistance levels.
- Two `SPSupportAgent`'s, with either 20% or 10% swing point identification threshold.
- Two `TrendLineAgent`'s, using either a 20% or 10% swing point identification threshold to find the points that (potentially) form trend lines.
- One `FibonacciAgent`, using a 20% swing point identification threshold.
- One `RSILevelAgent`, using a 14-day RSI.
- One `RSIDivergenceAgent`, also using a 14-day RSI.
- One `VolumeAgent`, with a percentage volume oscillator (PVO) configured with a 5- and 15-day volume moving average.

The hidden layer was configured to hold 8 neurons. The learning rate was set to 0.3 and the training momentum was set to 0.65.

7.3.2 Empirical Results

The 2-day ANN was tested with DNBNOR and FRO stock data. The results are given in tables 7.5 and 7.6

	Unfiltered			Filtered		
Year	ρ	$\overline{err}$	r	ρ	$\overline{err}$	r
2005	0.1268	0.01732	0.5905	-0.0005475	0.02471	0.5844
2006	0.1276	0.02285	0.5154	-0.1019	0.03676	0.5774
2007	0.1154	0.02081	0.5023	0.06694	0.02732	0.5300
2008	0.0511	0.03834	0.5557	0.07032	0.04205	0.5392

Table 7.5: Results for the 2-day ANN with DNBNOR stock data

	Unfiltered			Filtered		
Year	ρ	$\overline{err}$	r	ρ	$\overline{err}$	r
2005	0.02923	0.02481	0.5288	-0.04628	0.02437	0.5346
2006	0.04022	0.03581	0.5394	0.08343	0.04770	0.5647
2007	0.09320	0.04526	0.4438	-0.03201	0.05922	0.4479
2008	-0.008395	0.04454	0.5063	0.006213	0.06108	0.5439

Table 7.6: Results for the 2-day ANN with FRO stock data

7.4 The 1-Week Prediction ANN

7.4.1 Network Architecture

The 1-week ANN used a similar input layer architecture as the 2-day, with the agent parameters adjusted so that the agents took a slightly longer term view than before. The hidden layer size and the learning and moment rates where kept the same as for the 2-day.

7.4.2 Empirical Results

The 1-week ANN was tested with DNBNOR and JIN stock data. The results are given in tables 7.7 and 7.8.

	Unfiltered			Filtered		
Year	ρ	$\overline{err}$	r	ρ	$\overline{err}$	r
2005	0.1179	0.01496	0.606	0.3656	0.01712	0.5464
2006	0.1684	0.02102	0.46	0.1337	0.01462	0.5213
2007	0.01635	0.02129	0.4827	0.05191	0.02021	0.4871
2008	-0.01921	0.07115	0.5797	-0.01946	0.07525	0.5692

Table 7.7: Results for the 1-week ANN with DNBNOR stock data

	Unfiltered			Filtered		
Year	ρ	$\overline{err}$	r	ρ	$\overline{err}$	r
2005	0.03296	0.06152	0.5957	-0.05932	0.06204	0.5928
2006	0.1143	0.05301	0.5954	-0.05294	0.05207	0.6457
2007	0.08528	0.06136	0.5954	-0.0182	0.07047	0.4563
2008	0.04969	0.103	0.3663	-0.1082	0.06875	0.4104

Table 7.8: Results for the 1-week ANN with JIN stock data

Chapter 8

Discussion

On first inspection, we see that the results of the previous chapter are somewhat of a mixed bag. Let's consider each of the networks individually before making any general statements.

8.1 The 1-Week ANN

The first thing we can note is that the 1-week ANN performed significantly better on both stocks over 2005 and 2006 than it did over 2007 and 2008. In fact, this can be seen for all the ANNs. The main reason for this might be that the stock market in general had a major correction in the latter half of 2007, and underwent a period of extreme volatility which lasted through 2008. Since the networks were trained with data from a period of a relatively steady uptrend market (2005-2006), they had not been exposed to these types of market conditions, and so it would be reasonable to expect a somewhat lower performance in this period.

Looking at the results from the DnB NOR (DNBNOR) simulation in table 7.7, we see that there was a statistically significant positive bias on the correlation factors for the first two years. To have a correlation of nearly 0.4 on pure random luck, with $\sim 100,000$ forecasts recorded, seems extremely unlikely, particularly since it is significantly higher than zero in

Figure 8.1: Price chart of the JIN stock, 2005-2008

both those years. The hit-rate is also slightly higher than 50% on average, but not enough to call it a statistical significance.

Turning to the results from the somewhat less traded JIN stock (table 7.8) we see that performance was consistently lower here. The hit rate of just 36% in 2008 is indeed an abysmal result, but it can be understood better if we look at the actual price curve for JIN in that period, shown in fig. 8.1. Here we see that in late 2007, the stock went into a period of extreme volatility, going from over 90 NOK to less than 10 NOK in the following year. From our results, it's clear that the neural network was nowhere near prepared for such a scenario. In the more normal market conditions of 2005 and 2006, we see that the hit rate was good, averaging around 60%, but the low correlation factors could indicate that this was

more luck than skill on part of the ANN.

These results, in particular for DnB NOR, seem to indicate that the 1-week ANN does indeed have significant predictive power *during normal market conditions.*

8.2 The 2-Day ANN

The 2-day ANN performed relatively poorly in 2007 in particular, and 2008 to a lesser extent, in similar fashion to the 1-week ANN. This is probably for the same reasons that we just discussed.

In all of the 8 simulations, only one had hit rate less than 50% (FRO in 2007), which can be taken as a positive result. The fact that the unfiltered correlations are all over 0.1 for the first three years in table 7.5 here is also positive. Still, the correlation factors should've been somewhat higher if we were to make a definite positive conclusion here. It's interesting to note from table 7.5 that the correlation is a lot worse for the filtered values. It's hard to come up with a good explanation for why this might be.

The results from the Frontline simulations (table 7.6) are less encouraging than the DnB NOR ones (table 7.5). Here the correlations are all consistently close to zero, indicating that the network was unable to learn any useful features in the training. As can be seen by comparing figures 8.2 and 8.3, the FRO stock was generally a lot more volatile than the DNBNOR stock (except for maybe 2008), which may partly help explain this result.

8.3 The 2-Hour ANN

Looking at tables 7.3 and 7.4, we see that although the results tend slightly to the positive side, there correlations and hit rates are lower than what we were able to obtain for the weekly and 2-day ANNs. The average error is also significantly lower, but that is as expected when considering that the stock price variations in a two-hour time frame will generally be lower than on the weekly or 2-day time frame.

Figure 8.2: Price chart of the DNBNOR stock, 2005-2008

One indication that the network may have learned something useful for the 2005 and 2006 tests is that the performance here was consistently better than for the following two years, which is the same effect that was witnessed in the 2-day and weekly networks. If there had been no predictive power at all, and the predictions were fully uncorrelated to the curve, we wouldn't expect the performance to be negatively affected when exposed to abnormal data.

Still, while the results do tend more to the positive side than the negative (especially in 2005 and 2006), the statistical significance is regrettably too small to make a definite statement on whether the network was successfuly able to learn anything useful or not.

Figure 8.3: Price chart of the FRO stock, 2005-2008

8.4 The 30-Minute ANN

The 30-minute ANN had one particularly good test with DNBNOR in 2005 (table 7.2), which sticks out among the rest. This may of course just have been a lucky fluke, especially considering the abysmal performance of the test on the subsequent year, which has the worst hit rates witnessed in all the tests performed on any time frame. And apart from with this one test, correlations are generally close to zero. This indicates that the network was, at least for the most part, unable to learn anything useful at all.

8.5 Overall Performance

Reviewing the results, we see that ANN prediction performance on the longer time frames (1-week and 2-day) do show promise of being able to predict prices significantly better than what a random generator would, as long as market conditions remain normal. The 2-hour network also gave slightly positive results, but we may have wished to see somewhat higher correlations if we were to make any definite statements on its usefulness. The 30-minute network didn't show much promise of being able to reliably predict price fluctuations at all.

Chapter 9

Conclusion

9.1 Krang: Does It Work?

Based on the discussion of the previous chapter, it does indeed seem like at least some of the networks that were developed using Krang managed to learn enough useful features that they would reliably outperform a random prediction generator at least some of the time.

As such, these results add to the list of similar results reported by other researchers[9] that contradict the Efficient Market Hypothesis.

It should be noted that it has not been investigated how well the generated price predictions can be used as a basis for actual trading strategies. The generated predictions are only valuable from an investment standpoint if they help generate profits which reliably outperform just owning the stocks in terms of risk/reward (Sharpe) ratio. As such, the actual usefulness of the Krang system as an investment tool has not been sufficiently demonstrated.

And although some of the results were positive to a statisically significant degree, a lot of the results were less encouraging. The following sections review some factors that may help explain the mixed quality of the results, with some ideas for potential improvements that may help ameliorate these factors.

9.1.1 Adverse Market Conditions

While some of the networks did show statistically significant predictive performance in 2005 and 2006, they were generally unable to handle the stock market turmoil that occured in 2007 and 2008. This just illustrates one of the major inherent problems with applying pretty much any type of supervised machine learning method to financial time series prediction: it cannot be prepared for that which it has not already seen.

So while there may be recurring patterns and features in the time series that can be learned reliably by these networks, there is always a possibility that the market will start to act abnormally. For anyone thinking of deploying a system similar to Krang for real time live trading, it would therefore be important to implement proper safety mechanisms in case such market conditions should occur.

9.1.2 Long Term Vs. Short Term

It was observed that the longer term networks seemed to outperform the shorter term networks, as network performance became less and less promising on the shorter time scales.

It may be that more technical analysis methods would need to be implemented (as agents) that are more appropriate for short term analysis, because the majority of the agents that were implemented in Krang had to be discarded for the intraday networks. Intuitively, it's clear that e.g. a divergence in the 14-day RSI tells very little about what will happen in the next few minutes on the curve, so it seems that one might need to come up with methods that are more likely to capture short-term conditions to get better short-term performance (if it's possible at all to use an agent-based ANN for this purpose).

9.1.3 The Problem Of ANN Design

A major challenge in generating the results of this report has been to find good artificial neural network (ANN) architectures. There are so many variables that need to be configured, that finding a good set of settings

for each ANN becomes an overwhelming task in itself. First one needs to decide what agents to include, then how each agent should be configured, and then how many hidden layers to have and how many nodes to put in each of them. One also needs to find good values for the learning rate and momentum parameters that are used in the backpropagation algorithm. To exhaustively test for so many variables manually by trial and error is close to impossible, since each variable adds another dimension to the search space of possible ANN configurations and so increases it exponentially.

One idea for future improvement in this area might be to use some type of genetic algorithm, or perhaps a heuristic based on Netwon's method, to find good ANN configurations automatically prior to empirical testing.

9.1.4 Reversal Patterns

In the Krang system, only two agents were implemented for recognizing trend reversal patterns (the `DoubleTopAgent` and `DoubleBottomAgent`). Even though they only tested for the simplest possible patterns (the double top and double bottom), they were both unable to classify patterns with sufficient frequency to be of any use to the forecasting ANNs.

This is probably because the problem of doing automatic pattern recognition like this is very challenging to solve by making a conventional heuristic. As such, applying pattern recognition ANNs to identify trend reversal patterns could be a better way to solve this problem than the heuristic methods in the Krang system.

Trend reversal patterns are an important part of the technical analysis discipline, so it seems likely that having better methods for automatically identifying them in the Krang system could have increased ANN forecasting performance further than what was observed.

Index

Bibliography

[1] Aamodt, R. and Aase, K. The applications of machine learning techniques in financial analysis, December 2009.

[2] Berk, J. and DeMarzo, P. *Corporate Finance*. Pearson Education, 2007.

[3] Ellen Brown. Computerized front-running. *CounterPunch.org*, 2010.

[4] P. Cootner. *The random character of stock market prices*. MIT Press, 1964.

[5] R.N. Elliot. *The Wave Principle*. Collins, 1938.

[6] Floreano, D. and Mattiussi, C. *Bio-Inspired Artificial Intelligence*. MIT Press, 2008.

[7] Bank for International Settlements. Foreign exchange and derivatives market activity in 2007. `http://www.bis.org/publ/rpfxf07t.pdf`, 2008.

[8] D. Hebb. *The Organisation of Behavior*. Wiley, 1949.

[9] Huang, W., Lai, K.K., and Nakamori, Y. Forecasting foreign exchange rates with artificial neural networks: A review. *International Journal of Information Technology & Decision Making*, 3(1):145–165, 2004.

[10] Kelso, S., Ganong, A., and Brown, T. Hebbian synapses in hippocampus. *Proceedings of the National Academy of Sciences*, 83:5326–5330, 1986.

[11] J. Kramer. Commodity trading: History. `http://www.econoutlook.com/2008/07/commodity-trading-part-1-history.html`.

[12] F. Larsen. Automatic stock market trading based on technical analysis. Master's thesis, Norwegian University of Science and Technology (NTNU), 2007.

[13] Lo, A.W. and MacKinlay, A.C. *A Non-Random Walk Down Wall Street*. Princeton University Press, 1999.

[14] History of oslo børs. `http://www.oslobors.no/ob_eng/Oslo-Boers/About-us/The-history-of-Oslo-Boers`.

[15] R.R.Jr. Prechter, editor. *R.N. Elliots Market Letters*. New Classics Library, 1993.

[16] S.W. Roberts. Control chart tests based on geometric moving averages. *Technometrics*, 42(1):97–101, 2000.

[17] Rowley, H.A., Baluja, S., and Kanade, T. Neural network-based face detection. *IEEE Transactions On Pattern Analysis and Machine Intelligence*, 20:23–38, 1998.

[18] Samolada, E. and Zapranis, A. Can neural networks learn the head and shoulders technical analysis price pattern? towards a methodology for testing the efficient market hypothesis. *Lecture Notes in Computer Science*, 4669, 2007.

[19] M. Smith. *Neural networks for statistical modeling*. Van Nostrand Reinhold, 1993.

[20] Stockcharts.com chartschool. `http://stockcharts.com/school`.

[21] W3C. *Extensible Markup Language (XML) 1.0 Recommendation*, 5th edition, November 2008.

[22] S. Walczak. An empirical analysis of data requirements for financial forecasting with neural networks. *Journal of Management Information Systems*, 17(4):203–222, 2001.

[23] Walczak, S. and Cerpa, N. Heuristic principles for the design of artificial neural networks. *Information and Software Technology*, 41:107–117, 1999.

[24] Wikipedia: Amsterdam stock exchange. `http://en.wikipedia.org/wiki/Amsterdam_Stock_Exchange`.

[25] Wikipedia: Black monday (1987). `http://en.wikipedia.org/wiki/Black_Monday_(1987)`.

[26] Wikipedia: Commodity markets. `http://en.wikipedia.org/wiki/Commodity_markets`.

[27] Wikipedia: Financial market. `http://en.wikipedia.org/wiki/Financial_market`.

[28] Wikipedia: Security (finance). `http://en.wikipedia.org/wiki/Security_(finance)`.

[29] Wikipedia: Time series. `http://en.wikipedia.org/wiki/Time_series`.

[30] J.W. Wilder. *New Concepts in Technical Trading Systems*. Trend Research, 1978.

[31] Yahoo! finance. `http://finance.yahoo.com`.