An empirical analysis of algorithmic trading on financial markets

Masterproef voorgedragen tot het bekomen van de graad van:
Master in de Bedrijfseconomie

Reinout Declerck

onder leiding van:
Prof. dr. Michael Frömmel
An empirical analysis of algorithmic trading on financial markets

Masterproef voorgedragen tot het bekomen van de graad van:

Master in de Bedrijfseconomie

Reinout Declerck

onder leiding van:

Prof. dr. Michael Frömmel
PERMISSION

Ondergetekende verklaart dat de inhoud van deze masterproef mag geraadpleegd en/of gereproduceerd worden, mits bronvermelding.

Reinout Declerck
The present work comes at the final stage of eight years of university studies, and new experiences and new challenges are waiting around the corner. I am first and foremost grateful to Prof. dr. Michael Frömmel for giving me the opportunity to undertake a study in the specific field of algorithmic trading, and for encouraging me and helping me out along the way. In addition, I am greatly indebted to my friends and family for their unconditional support.

Reinout Declerck
May 18th 2008
# Contents

**List of Figures**  xi  
**List of Tables**  xiii  

1 **Introduction**  1  

2 **The nature of technical analysis and technical trading strategies**  3  
   2.1 Genetic algorithms applied to technical trading strategies  13  
   2.2 Artificial neural networks applied to technical trading strategies  15  

3 **Testing the technical trading strategies**  17  
   3.1 Common testing errors (and some of their solutions)  17  
   3.2 Why can it work?  19  
      3.2.1 Theoretical explanations  20  
      3.2.2 Empirical explanations  21  
   3.3 Trading agents in more realistic circumstances  23  

4 **An advanced trading agent in action**  27  
   4.1 The adaptive reinforcement learning trading agent  27
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.2</td>
<td>Some remarks on the implementation</td>
<td>35</td>
</tr>
<tr>
<td>4.3</td>
<td>Preliminary testing of the ARL trading agent</td>
<td>36</td>
</tr>
<tr>
<td>4.4</td>
<td>The ARL trading agent put to the real test</td>
<td>43</td>
</tr>
<tr>
<td>4.4.1</td>
<td>The data</td>
<td>43</td>
</tr>
<tr>
<td>4.4.2</td>
<td>The results</td>
<td>45</td>
</tr>
<tr>
<td>5</td>
<td>Conclusions</td>
<td>51</td>
</tr>
<tr>
<td>A</td>
<td>Source code of the ARL trading agent</td>
<td>53</td>
</tr>
<tr>
<td></td>
<td>Bibliography</td>
<td>65</td>
</tr>
</tbody>
</table>
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Number of studies dedicated to technical trading in the period 1960-2004. The source data for this graph was collected from Park and Irwin [2007].</td>
</tr>
<tr>
<td>4.1</td>
<td>The ARL trading agent in action.</td>
</tr>
<tr>
<td>4.2</td>
<td>Illustration of the primitive data set, consisting of 16 returns (expressed in units of pips).</td>
</tr>
<tr>
<td>4.3</td>
<td>Mean percentage of the theoretical maximal gain for 10 consecutive runs, with respect to the number of weights $w$. Error bars denote the standard deviation from the mean. $\delta = 0.0$, $\eta = 0.2$, $\rho = 0.05$, $x = 100.0$, and $y = 0.0$. Out-of-sample data points = 4000 (hence, the theoretical maximal gain equals 4000 pips), re-training of the neural network every 500 data points, using 2000 data points of past returns. The number of training epochs was 10.</td>
</tr>
</tbody>
</table>
4.4 Best value for the utility function $U_{4000}$ (4.15) in five consecutive iterations of optimizations in all five parameters with respect to the number of weights $w$. Starting parameters are $\delta = 0.0$, $\eta = 0.2$, $\rho = 0.05$, $x = 100.0$, and $y = 0.0$. Out-of-sample data points = 4000, re-training of the neural network every 500 data points, using 2000 data points of past returns. The number of training epochs was 10. The red line depicts the maximal theoretical value of the utility function for 4000 out-of-sample data points.

4.5 The Euro-dollar exchange rate (01/10/06 - 31/12/07). The source data for this graph was collected from Dukascopy’s CSV data export.

4.6 Cumulative returns as a function of time for three different numbers of weights. The details of this simulation have already been described in Table 4.2. Note that the first 10000 points are used for initial optimization of the ARL trading agent, therefore no profits are generated there. For visual reference only (since the scale on the $y$-axis is incorrect), we have reprinted (in light-gray) the euro-dollar exchange rates from Figure 4.5.
List of Tables

4.1 Boundaries of the five-dimensional parameter space . . . . . . 41
4.2 Profits generated by the ARL trading agent using the euro-
dollar exchange rate data of Figure 4.5. Starting parameters
are $\delta = 0.0$ (if not constrained to a fixed value), $\eta = 0.2$,
$\rho = 0.05$, $x = 100.0$, and $y = 0.0$. Re-training of the neural
network every 500 data points for 10 training epochs, using
2000 data points of past returns. (Re-) optimization of all five
(four if $\delta$ is constrained) parameters every 10000 data points for
five iterations. Optimizations were performed in the parameter
space defined in Table 4.1 nc = not counted. . . . . . . . . . . . 47
4.3 Percentage gain and percentage deviation for 6 simulations of both a constraint-free ($\delta = 0$) and on a constrained ($\delta = 0.5$) 16-weights ARL trading agent using the euro-dollar exchange rate data of Figure 4.5. Other starting parameters are $\eta = 0.2$, $\rho = 0.05$, $x = 100.0$, and $y = 0.0$. Re-training of the neural network every 500 data points for 10 training epochs, using 2000 data points of past returns. (Re-) optimization of all five (four if $\delta$ is constrained) parameters every 10000 data points for five iterations. Optimizations were performed in the parameter space defined in Table 4.1.  nc = not counted. . . . . . . . . . . . 49
In financial markets, algorithmic or automated trading (AT) denotes the use of computer software for entering trading orders with computer algorithms managing certain aspects of the order such as the timing, price, and/or quantity. The use of AT has evolved simultaneously with the steep rise in numerical computing power: from nearly zero about a decade ago, AT was believed to account for a third of all European Union and United States stock trades in 2006 (with some specialised markets featuring a much higher proportion), and by 2010, it is estimated that approximately 53% of all equities trading worldwide will be done through some form of algorithmic trading. [Aite (2006); Economist (2007)] Applications range from dividing up large trades into several smaller trades in order to minimize market impact, to the (more ambitious) forecasting of future (price) trends and generation of trading decisions based upon the outcome of these forecasts. This dissertation focuses on the latter, and we will refer to them in this work as agents, as this group of trading algorithms is – in principal – designed for autonomous operation. The controversy on the profitability of these agents has sparked a lively discussion among scientists for quite some time now, and this is the
Introduction

very reason why we will devote this dissertation to it. We will attempt to get insight into the profitability of trading agents, both by consulting the available scientific literature for accounts on the subject, and by performing a thorough study of a fairly advanced trading agent which was recently described in literature.

Trading agents predominantly rely on underlying technical analysis principles, and therefore, we will start this dissertation by familiarizing ourselves with the nature of technical indicators and technical trading strategies in Chapter 2. Chapter 3 is devoted to a literature review covering on one hand the common testing errors (and some of their solutions) in assessing the performance of technical trading strategies, and on the other hand the various proposed theoretical and empirical explanations for the profitability of technical trading.

The second – and most important – part of this dissertation (Chapter 4) involves the study of the adaptive reinforcement learning trading agent introduced by Dempster and Leemans (2006). Based on this actual and hands-on experience, we wish to gain a clearer understanding of what is possible within the boundaries of today’s publicly available technology. A significant part of this work involved the translation of the algorithms from Dempster and Leemans (2006) into computer code (which is included as supplementary information in Appendix A). Subsequently, the performance of the study trading agent is submitted to a careful investigation. The trading agent will be trained, applied, and evaluated on intraday historical euro-dollar foreign exchange rates.

1 This is the trading frequency at which technical trading strategies are most commonly used.
Technical trading is based on technical analysis, a method of forecasting price movements using past market data of a security (e.g. stocks, bonds, currencies, or property) such as price, volume, ...

In contrast to the views of its practitioners, many academic economists have been and remain critical towards technical analysis, mainly because of its clear contradiction with the efficient market hypothesis. As illustrated in Figure 2.1, this controversy...

1 Approaches to forecasting the future direction of a security’s price fall broadly into two categories, i) those that rely on technical analysis, and ii) those that rely on fundamental analysis. While technical analysis uses only historical data (past prices, volume, ...) to determine the movement in the price of a security, fundamental analysis is based on external information, such as interest rates, prices and returns of other securities, and many other macro- or micro-economic variables.

2 The efficient market hypothesis asserts that financial markets are informationally efficient, or that prices on traded securities already reflect all known information and therefore are unbiased.
The nature of technical analysis and technical trading strategies

has resulted in an abundant literature during the last fifty years, sometimes of disputable scientific value in the early days (see Chapter 3) and with a significant higher number of studies since halfway the nineties, for the purpose of either uncovering profitable trading rules or testing the efficient market hypothesis, or both. No matter what the outcome of the scientific discussion, technical analysis is enjoying a wide-spread use: all major stock brokers publish technical commentary on the market and individual securities, and many of the columns and newsletters published by various (sometimes self-declared) experts are based on technical analysis. This apparent popularity alone provides already a stimulus to learn more about the phenomenon.

Technical analysis is based on the following three principles (the formulation below is found abundantly on the internet, eg. on http://www.alpari-idc.com/en/market-analysis-guide/technical-analysis/overview.html. However, no original source could be found):

- **Price discounts everything.** Price is affected by economic, political and other factors, and all information is already reflected in it. Technical analysis utilizes the information captured by the price to interpret what the market is saying with the purpose of forming a view on the future.

- **Price movements are not totally random, or prices move in trends.** The main purpose of the charts is to define a trend at an early stage and to trade in accordance with its direction.

- **History tends to repeat itself.** The techniques which were effective in the past can still be effective to forecast future price movements.

in the sense that they reflect the collective beliefs of all involved parties about future prospects. Therefore, these securities always trade at their fair value, making it impossible for investors to either purchase undervalued stocks or sell stocks for inflated prices. [Fama (1965)]
Figure 2.1: Number of studies dedicated to technical trading in the period 1960-2004. The source data for this graph was collected from Park and Irwin (2007).
The enormous amount of raw market data generated in the trading of a security is generally very difficult to process by the human brain. Therefore, numerous technical indicators have been developed over the years to highlight particular aspects of a security’s past performance, and thereby assist technical analysts in detecting possible future price movements. In order to get a better understanding of the nature of these indicators (and in particular how conceptionally uncomplicated they sometimes can be), it is instructive to describe (non-exhaustively) some of the most popular examples. These indicators (and their associated strategies, see below) appear to be common knowledge, though, and authorship is sometimes hard to retrieve, therefore we are only able to acknowledge the Tradetrek.com website, on which many of these indicators and strategies are explained:

- **Moving average**: a security’s price/time series can be seen as the superposition of a long-term trend and short-term, randomly fluctuating noise. In order to obtain a clean trend signal, short-term noise can be filtered out by using moving averages. The simple $n$-interval (prior-) moving-average time series $\text{SMA}_n$ is defined as:

$$\text{SMA}_n[p](t_i) = \frac{1}{n} \sum_{j=i-n+1}^{i} p(t_j), \quad (2.1)$$

where $p(t_i)$ denotes the price of the security at time $t_i$. The simple moving average defined above assigns equal weights to every point in the averaging interval; consequently, it may not emphasize the most recent price behaviour. To overcome this, one might consider using a linearly weighted $\text{WMA}_n$ or an exponential moving average $\text{EMA}_\alpha$ instead, defined respectively as:

---

3http://www.tradetrek.com/Education/default.asp
The nature of technical analysis and technical trading strategies

\[ WMA_n[p](t_i) = \frac{\sum_{j=i-n+1}^{i}(n + i - j)p(t_j)}{n(n+1)} , \quad (2.2) \]

\[ EMA_\alpha[p](t_i) = \frac{\sum_{j=1}^{i}(1 - \alpha)^{i-j}p(t_j)}{\sum_{j=1}^{i}(1 - \alpha)^{i-j}} , \quad (2.3) \]

with \( 0 < \alpha < 1 \).

- **Bollinger Band**: named after J. Bollinger, Bollinger (2001)] Bollinger bands serve two primary functions: i) to identify periods of high and low volatility, ii) to identify periods when prices are at extreme, and possibly unsustainable, levels. The Bollinger band includes 3 lines: the center line (C), the upper band (U), and the lower band (L). The center line is the simple \( n \)-interval moving average of a security’s price/time series, and the upper and lower bands are defined as, respectively, the center line plus/minus \( m \) times the standard deviation of that security in that \( n \)-interval:

\[ C_n[p](t_i) = SMA_n[p](t_i) , \]

\[ U_{m,n}[p](t_i) = SMA_n[p](t_i) + m \sqrt{\frac{1}{n} \sum_{j=i-n+1}^{i} (p(t_j) - SMA_n[p](t_i))^2} , \]

\[ L_{m,n}[p](t_i) = SMA_n[p](t_i) - m \sqrt{\frac{1}{n} \sum_{j=i-n+1}^{i} (p(t_j) - SMA_n[p](t_i))^2} . \]

(2.4)

The standard deviation ensures that the bands will react quickly to price movements and reflect periods of high and low volatility. Sharp price increases (or decreases), and hence high volatility, will lead to a widening
of the bands, whereas during a period of low volatility, the distance between the two bands will contract. Crossings of the upper and lower bands reflect relative overbought and oversold situations, respectively.

• **Support and resistance levels**: a support level is a price level where the price of a security tends to find support as it is going down. This means that the price is more likely to bounce off this level rather than break through it. However, once the price passes beyond this level, the price will likely continue dropping until a new support level is found. A resistance level has exactly the opposite features of a support level.

• **Relative strength index**: the relative strength index (RSI) compares the magnitude of a security’s recent gains to the magnitude of its recent losses and turns that information into a number that ranges from 0 to 100.

For each time step, an upward change $U$ or downward change $D$ amount is calculated. For a price increase, i.e. the price at $t_i$ is higher than $t_{i-1}$, we define:

$$U[p](t_i) = p(t_i) - p(t_{i-1})$$
$$D[p](t_i) = 0 ,$$

(2.5)

or conversely, on a price decrease (note that $D$ is a positive number):

$$U[p](t_i) = 0$$
$$D[p](t_i) = p(t_{i-1}) - p(t_i) .$$

(2.6)

The RSI is then calculated as:
The nature of technical analysis and technical trading strategies

\[
RSI_k[p](t_i) = 100 \frac{EMA_\alpha[U[p]](t_i)}{EMA_\alpha[U[p]](t_i) + EMA_\alpha[D[p]](t_i)} . \quad (2.7)
\]

- **Money flow index**: the money flow index (MFI) is an indicator that is similar to the RSI in both interpretation and calculation. However, the MFI is a more trustworthy indicator in the sense that it includes the volume \(v\) as a weighting factor, and is therefore a good measure of the strength of money flowing in and out of a security.

For each time step, an upward change \(U\) or downward change \(D\) amount is calculated. On a price increase, i.e. when the price at \(t_i\) is higher than \(t_{i-1}\), we define:

\[
U[p,v](t_i) = p(t_i)v(t_i) \\
D[p,v](t_i) = 0 , \quad (2.8)
\]

or conversely, on a price decrease (note that \(D\) is a positive number):

\[
U[p,v](t_i) = 0 \\
D[p,v](t_i) = p(t_i)v(t_i) . \quad (2.9)
\]

The MFI is then calculated as:

\[
MFI[p,v](t_i) = 100 \frac{U[p,v](t_i)}{U[p,v](t_i) + D[p,v](t_i)} . \quad (2.10)
\]

Using these and (an extensive amount of) similar technical indicators, trading strategies have been developed over the years, in which trading directives are generated based on the outcome of one or more of these indicators. A limited
The nature of technical analysis and technical trading strategies

set of some simple and quite common strategies is given below, mainly to illustrate the ideas underlying their creation:

- **Moving-average oscillator**: buy and sell signals are generated by two moving averages (SMA, WMA, or EMA), calculated using respectively a longer \( n_l \) and a shorter interval \( n_s \). In its simplest form, this strategy results in buying (or selling) when the shorter-interval moving average rises above (or falls below) the longer-interval moving average. When the shorter-interval moving average penetrates the longer-interval moving average, a trend is considered to be initiated.

The moving-average convergence/divergence (MACD), a more advanced trend-change indicator, is calculated as the difference between a fast \( \alpha_f \) and a slow \( \alpha_s \) EMA (which usually take into account the closing prices of the last 12 and 26 trading days, respectively),

\[
\text{MACD}_{\alpha_f,\alpha_s}[p](t_i) = \text{EMA}_{\alpha_f}[p](t_i) - \text{EMA}_{\alpha_s}[p](t_i),
\]

and is then compared against a signal line by smoothing this MACD with a further EMA (with \( \alpha_{sl} \), often chosen to include closing prices of the last 9 trading days):

\[
\text{MACD}^{\text{smooth}_{\alpha_f}}_{\alpha_f,\alpha_{sl}}[p](t_i) = \text{EMA}_{\alpha_f}[\text{MACD}_{\alpha_f,\alpha_s}[p](t_i)].
\]

Trading signals generated by the MACD include for example i) the MACD line crossing the signal line, or ii) the MACD line crossing zero. The former is the most commonly used trading signal, i.e. buying when the MACD crosses up through the signal line, or selling when it crosses down through the signal line. A crossing of the MACD line up through
zero is interpreted as bullish, or down through zero as bearish. These crossings are of course simply the fast EMA line crossing up or down through the slow EMA line. The MACD is considered most effective in times of widely swinging market situations.

- **Range break-out**: in this strategy, trading signals are generated as security prices hit new highs or new lows. A buy signal is generated when the price penetrates the resistance level. At this level, many investors are willing to sell, and this selling pressure will cause resistance to a price rise. It is assumed that when the price rises above the resistance level, it has broken through this level. Such a breakout is considered to be a buy signal. A sell signal is generated when the price penetrates the support level. The underlying idea is that the price will not penetrate through the support level because investors are ready to buy (again) at this price. However, if the price goes below the support level, it is to be expected that the price will drift further downward.

- **RSI/MFI-based strategy**: a security is overbought if the RSI/MFI reaches a given upper limit (normally in the 70-80 range), meaning that the investor should consider selling. The opposite holds true for a given lower limit (normally in the 20-30 range), at which point the investor should consider buying. The principle is that when there is a high proportion of daily movement in one direction it suggests an extreme, and prices are likely to reverse.

- **Combination strategy**: combination strategies are based on the security’s price (and traded volumes) satisfying a combination of conditions of various technical indicators. In the stochastic combination trading strategy, for example, a buy signal is generated when the security is in a long-term up trend, the momentum indicators indicate that the stock
The nature of technical analysis and technical trading strategies

is oversold, the RSI is low, and the price is near the lower Bollinger
Band, showing the initial signs of a bounce-back. For the sell signal, all
indicators will have to show the opposite signals.

The trading strategies described in the previous overview are illustrative not
only for a broader collection of strategies, but also generally for their very much
ad-hoc specification: a few often simplifying and biasing (in favour of easily
comprehensible dependencies preferred by human traders) assumptions about
the behaviour of a security are adopted ex ante (often made plausible only by a
typical example where these assumptions hold), and on this basis trading rules
are derived. It thereby remains unclear if the adopted assumptions are valid
more generally, or – expressed in a statistically more correct way – if the gains,
generated in the cases where the assumptions hold, more than compensate
the losses generated in those cases where the assumptions fail. If we wish to
genuinely evaluate the performance of technical trading, we must first of all
consider a broader, ideally assumption-free set (or space) of technical trading
strategies, in which we seek to find the optimal trading rules for a given
security. We will postpone, for now, the more general discussion on whether
or not technical analysis can be profitable at all, and concentrate first on the
available methods to circumvent the distorted picture of technical trading that
is introduced if only ex-ante trading rules are considered in its evaluation.

There exist two popular classes of methods to find the optimal trading rules.
In the first class of methods, an extensive selection of candidate trading rules
is gathered. Among these strategies, the optimal one (with respect to eg. the
(risk-adjusted) returns) for a given training period of a security’s price/time
series (the data set) can be derived using search algorithms such as for example
genetic algorithms. A genetic algorithm is a technique used in computing
to find exact or approximate solutions to optimization and search problems.
 Genetic algorithms uses techniques inspired by evolutionary biology such as inheritance, mutation, selection, and crossover. In the second class of methods, trading rules are generated by an artificial neural network which was optimized for a given security. An artificial neural network is a non-linear statistical data modeling tool based on biological neural networks, which can be used to model complex relationships between inputs and outputs. The power and advantage of a neural network lies in its ability to represent both linear and non-linear relationships and in its ability to learn these relationships directly from the data being modeled.

2.1 Genetic algorithms applied to technical trading strategies

Genetic algorithms constitute a class of search, adaptation, and optimization techniques based on the principles of evolutionary biology. Genetic algorithms became popular starting with the work of Holland (1975). An evolutionary algorithm submits a population of solution candidates to a fitness test, through the evaluation of each solution candidate to a fitness function specifically defined for the particular problem. New solution candidates are created by selecting relatively fit members of the population and recombining them through various operators, including a mutation operator which serves to maintain genetic diversity when going from one generation of a population to the next. Genetic algorithms are superior over gradient ascent methods when the fitness function is not well-behaved (i.e. not continuous and/or not differentiable) or features several local maxima. However, these algorithms generally come at an elevated computational cost (with respect to eg. gradient ascent methods, or special-purpose analytical algorithms), because of the
2.1. Genetic algorithms applied to technical trading strategies

numerous evaluations of the fitness function that are needed. Moreover, genetic algorithms are not guaranteed to converge, and therefore it may be necessary to switch to one of the gradient ascent methods during the final steps. One final issue one must take care of is the problem of overfitting or overtraining, which can occur if the training data set (used to evaluate the fitness function) is too small in comparison with the amount of parameters in the solution candidates. These complex candidates are capable of reproducing the training data set very accurately, but have passed beyond the point of picking up patterns that can be generalized beyond the training data set, and thus contain generally very little predictive power.

In [Allen and Karjalainen (1999)], genetic algorithms were used to learn optimal technical trading rules for the Standard and Poor’s 500 index daily data from 1928 to 1995, rather than having them specified ex ante. Trading rules are composed of building blocks of simple functions of past price data, numerical and logical constants, and logical functions that combine lower-level building blocks. The uppermost building block is terminated by a Boolean function, generating either a buy (true) or a sell (false) signal. The fitness function is calculated as the excess return over a simple buy-and-hold strategy during a training period. Despite the completely unbiased way in which the trading rules were generated, no rules were found that generated consistent excess returns over the buy-and-hold strategy as soon as transaction costs were taken into account.
2.2 Artificial neural networks applied to technical trading strategies

Artificial neural networks (ANN) are interconnected networks of simple processing units (called nodes), which can be trained to model complex relationships between inputs and outputs of the network (much like the brain does), such as for example price/time series of a security at the input, and trading signals (buy or sell) at the output. The feedforward neural network is conceptually the most simple type of ANN, as information flows in only one direction (i.e. forward) from the input nodes, over layers of hidden nodes, to the output nodes. A simple extension to this type of ANN is the recurrent feedforward neural network, in which past output signals are redirected back to an input node. The inclusion of this recurrency generally improves the performance (i.e. the generalization ability) of the ANN. Each node processes signals in the following way:

\[ y = f \left( \sum_{j=0}^{n} w_j x_j + v \right), \]  

(2.13)

where \( \{x_j\} \) represents a number of input signals, corresponding with output signals of the previous layer, which are multiplied by a set of corresponding weights \( \{w_j\} \), \( v \) is a threshold value and \( f \) a predefined signal function, such as eg. the tanh function. \( y \) is the output signal that the node presents to the next layer. Through the adjustment of the weights \( \{w_j\} \) and the threshold \( v \) of each node, an ANN is trained to optimize a problem-specific cost function (calculated on a training data set of inputs and outputs), which essentially has the opposite meaning of the fitness function of a genetic algorithm. This cost function may feature several local minima, and therefore, genetic algorithms
2.2. Artificial neural networks applied to technical trading strategies

can be necessary here, too, next to the different gradient descent methods. Given their vastly increasing complexity for larger number of nodes, ANNs are also prone to the problem of overtraining or overfitting (see 2.1). Also, given an ANN optimized to model a specific training data set, there is generally no easy way to explain why exactly this configuration of the ANN is performing so well.

Park and Irwin (2007) find that studies in which ANNs have been applied to financial securities generally provide a positive outcome on the usefulness of technical trading strategies. These ANNs often took as input a combination of past returns and/or past trading signals, and returned at the output either trading signals or future price predictions. Amongst others, we wish to mention the extensive work of Gençay (1998a, b, 1999), Gençay and Stengos (1997, 1998) on daily observations of the Dow Jones industrial average index from 1963 to 1988, in which it was observed that technical trading rules based on ANN models outperform a simple buy-and-hold strategy, even when transaction costs and risk are taken into account. In the same spirit, Fernández-Rodríguez et al. (2000) applied an ANN to the Madrid stock market index, using daily data from January 2nd 1966 to October 12th 1997, and observed that their ANN outperformed the buy-and-hold strategy, although transaction costs were not included.
3 Testing the technical trading strategies

In this chapter we will review the available knowledge about the profitability of technical trading strategies, more specifically we will discuss the common errors (and its remedies) in testing a strategy, and the theoretical and empirical explanations that support generating profits through technical trading. Our primary resource here will be a recent and lengthy review paper by Park and Irwin (2007), supplemented with some other recent resources.

3.1 Common testing errors (and some of their solutions)

The controversy about the profitability of technical trading has sparked intense discussion among scientists for almost half a century (see Section 2). Many of the earlier studies, however, whether the outcome be favourable or unfavourable for technical trading, lacked the proper testing procedures to provide conclusive evidence on the (un-)profitability of technical trading. Some of the frequently occurring errors identified by Park and Irwin (2007)
3.1. Common testing errors (and some of their solutions) include:

- **Small number of strategies**: Many of the earlier studies considered only a limited number of strategies, typically one or two, and subsequently generalized the findings for these particular strategies to technical trading as a whole. However, while some strategies may indeed be unable to generate significant profits, this does not necessarily imply the failure of other (e.g., more advanced) strategies.

- **Absence of statistical significance tests**: A technical trading return is called statistically significant if it is unlikely to have occurred by chance. Although several studies conducted some statistical hypothesis test to measure the likelihood of their experimental results to cast doubt on the null hypothesis\(^1\) of an efficient market, the distribution of returns under the efficient market hypothesis is in fact not known, hence limiting the credibility of any conclusions drawn.

- **No consideration of the risk-return trade-off**: According to the risk-return trade-off principle, the potential return of an investment increases with the risk associated with that investment (i.e., low risk levels have low potential returns, whereas high risk levels have high potential returns). Thus, large trading returns may be nothing more than the reflection of greater risks, rather than the refutation of the efficient market hypothesis.

- **Awkward presentation of results**: Some early studies are difficult to interpret since they report the average performance of a set of trading rules or securities rather than the performance of individual trading

---

\(^1\) A null hypothesis is a hypothesis which needs to be refuted in order to support an alternative hypothesis.
strategies. This practice may conceal the fact that some trading strategies applied to particular securities are in fact profitable.

- **Data snooping bias**: Data snooping bias is caused by the misuse of the training data (i.e., past prices, volume, etc.) used to assess a trading strategy. If one just evaluates enough trading strategies, it is likely that eventually a profitable strategy for this particular data set will be found. However, this does not necessarily mean that this strategy will also be profitable when put into practice.

Modern studies have improved upon the aforementioned shortcomings of earlier studies, most notably since the work of Lukac and co-workers at the end of the eighties. Improvements include amongst others that generally more strategies have been considered in the study, that the data-snooping problem of apparent successful strategies has been addressed e.g., through out-of-sample verification or through the bootstrap reality check methodology of White (2000), and that statistical tests have been conducted to measure the significance of trading returns.

### 3.2 Why can it work?

In the last decade, a considerable amount of scientific evidence (albeit not yet conclusive) has appeared, indicating that the efficient market hypothesis may not hold in all markets and/or at all times. Rather than discussing the different measurement procedures in assessing these apparent flaws in the market efficiency, we opted to discuss the various proposed theoretical and empirical explanations on *why* exactly this may be the case. Here follows an overview of the prevailing (sometimes still controversial) views identified by
3.2. Why can it work?

Park and Irwin (2007). It is worthwhile noting that Menkhoff and Taylor (2007) also provide an overview of possible explanations, but their account is focused more exclusively towards technical trading in foreign exchange markets.

3.2.1 Theoretical explanations

Theoretical models which add ground for technical trading (and hence contradict the efficient market hypothesis) include i) noise in equilibrium prices, ii) behaviour models, iii) herd behaviour, iv) market power, and v) chaos:

- **Noise in equilibrium prices**: The noise model for equilibrium prices states that prices of traded securities adjust only slowly to new available information, whereas the efficient market hypothesis presupposes that all new information is readily reflected in the price. The most significant work on these noise models was carried out by Grossman and Stiglitz (1976, 1980), who basically argued that in an efficient market, no one has an interest in arbitrage (i.e. the correction of mispricings in the market), which is one of the cornerstones of the efficient market hypothesis, and as a result, the market will no longer be efficient. In other words, efficient markets are condemned to losing their efficiency.

- **Behaviour models**: In behaviour models, two types of investors are identified: noise traders, who trade based on irrational beliefs and sentiments (e.g. trend following, buying when prices rise and selling when prices drop, ...), and arbitrageurs, who invest based on rational expectations about the return of a security. Behaviour models propose that arbitrage (which is supposed to correct for any mispricings due to noise traders) by the latter group is limited because of the risk that the irrational market situation will continue. Rather, it may be more
3.2. Why can it work?

profitable for the arbitrageurs to mimic the behaviour of the noise traders for a while. Therefore, although arbitrageurs ultimately bring prices back to fundamental levels, in the short run they amplify the effect of noise traders, hence creating a limited time span during which the efficient market hypothesis fails. [De Long et al. (1990a,b)]

- **Herd behaviour**: Much along the lines of the behaviour models, herd behaviour (i.e. a group of individuals acting together without prior planned direction), can explain short-term irrational market behaviour. Technical traders in particular are trained to detect such an event in an early phase, allowing them to generate profits on it. Moreover, provided the group of traders which rely on technical analysis is large enough, they can even be the very cause and/or the reinforcing factor of such herd behaviour. [Froot et al. (1992)]

- **Chaos theory**: Chaos theory involves the study of deterministic non-linear dynamical systems, which can yield highly complex random-looking (but nonetheless deterministic) paths as a function of time. Clyde and Osler (1997) simulated such a chaotic time series and showed that i) technical trading strategies applied to such a series feature more predictive power than when applied to random data, and ii) technical trading strategies applied to such a series outperform a random trading rule.

3.2.2 Empirical explanations

Among the empirical explanations for the profitability of technical trading we distinguish i) intervention of central banks, ii) order flow, iii) temporary market
3.2. Why can it work?

inefficiencies, and iv) risk premiums.\(^2\)

- **Intervention of central banks**: Central banks intervene in foreign exchange markets in order to prevent volatility (or worse: shocks, something which can have consequences reaching far beyond the exchange rate itself). Essentially they buy currency in times of selling pressure, and vice versa, only to make the adjustment of the exchange rate to its new equilibrium as smoothly as possible. During this delayed transition, a trend arises, which technical traders can pick up and exploit to generate profits. [Saacke (2002)]

- **Order flow**: Two well-known technical trading predictions are supported by the analysis of Osler (2003): i) Down-trends and up-trends reverse course at pre-identifiable support and resistance levels, which are often round numbers, and ii) trends tend to be relatively fast after crossings of the support and resistance levels (see Chapter 2 on support and resistance levels and range break-out). Through analysis of orders placed at a large bank, Osler discovered that executed take-profit orders cluster more strongly at round numbers than executed stop-loss orders, whereas executed stop-loss buy orders cluster most strongly just above round numbers, and executed stop-loss sell orders cluster most strongly just below round numbers. Take-profit order tend to cause trend reversal, whereas stop-loss orders result in trend propagation.

- **Temporary market inefficiencies**: Several studies [eg. Olson (2004); Sullivan et al. (1999, 2003)] have indicated the profitability of technical trading before the nineties, followed by a considerable decline since then.

\(^2\)Additionally, [Park and Irwin (2007)] discuss the influences of market microstructure deficiencies and data snooping, but strangely both of these aspects seem to refute the possibility of generating profits through technical trading.
3.3 Trading agents in more realistic circumstances

There exist two possible explanations for this decline. On one hand, there is the self-destructive aspect of technical trading strategies: once these strategies get more widely adopted, their inherent reasons for success tend to get incorporated into prices, and as a result, these strategies will no longer be successful. Timmermann and Granger (2004). On the other hand, there are the structural changes of markets since the nineties, with the advent of fast electronic trading and increased liquidity (due to a general democratization of security trading), both of which will have contributed to the speed of price movements of securities, and in turn, a reduced profitability of technical trading strategies. Sullivan et al. (1999).

• Risk premiums: Some people argue that technical trading excess profits over a buy-and-hold strategy constitute merely a compensation for bearing additional risk. However, several studies, mainly those focused on foreign exchange and futures markets [eg. Chang and Osler (1999); LeBaron (1999); Lukac and Borsen (1990)], have found higher Sharpe ratios for technical trading strategies than for a buy-and-hold strategy. The Sharpe ratio provides a measure for risk-adjusted return, and will be further discussed in Section 4.1.

3.3 Trading agents in more realistic circumstances

Although many of the trading strategies proposed in literature have been extensively tested in a controlled environment, their profitability in real trading is subject to further uncertainty factors, such as for example a possible lag between the trading decision and the actual trade and/or a different price than planned by the agent (eg. because there were no buyers/sellers at the planned price).
3.3. Trading agents in more realistic circumstances

Putting a set of newly proposed trading agents to a live test would turn out to be a rather risky affair, as real money is being traded. Indeed, many major stock brokers possess a safe so-called backtesting environment, on which new trading strategies are carefully tested before going live. Unfortunately, there exists only little publicized information on this proprietary research. There do exist some public efforts in this field, such as the Automated Trading Championship[^3], but again, financial issues prevent any serious research to be carried out here: the organizing company is merely promoting its main product (a programming language), and most participants are unwilling to disclose their strategy, once they learn that it is being profitable.

One project that serves the sole purpose of scientific research is the Penn-Lehman automated trading project.[^Kearns and Ortiz (2003)] Albeit being sponsored by Lehman Brothers’ Proprietary Trading Group, i.e. a private company, it provides the most realistic and thorough account of automated trading strategies so far. The Penn-Lehman automated trading (PLAT) simulator performs a market simulation that (more or less) integrates the actions of the trading agents with real-world trades. The PLAT simulator extracts this real-world market data from an Electronic Communication Network (ECN), which is an electronic trading system that matches buy and sell orders automatically. The PLAT simulator merges the order book data from a particular ECN[^4] with orders placed by the trading agents, hence adding a new level of realism by creating a virtual ECN in which different trading agents can trade with the outside market and with each other.

Understanding the way an ECN functions requires the notion of limit orders and market orders[^5]. An ECN maintains two queues, called the buy and sell

[^3]: http://championship.mql4.com/
[^4]: The Island ECN, now owned by Instinet: http://www.instinet.com/
[^5]: A limit order is an order to buy or sell a security at a specified price. A market order is an
3.3. Trading agents in more realistic circumstances

order book, respectively, which contains limit orders ordered by price, with the highest offered unexecuted buy price at the top of the buy order book, and the lowest offered unexecuted sell price at the top of the sell order book. If there are multiple limit orders at the same price, they are ordered by their time of arrival, with the older orders higher in the book. When a new order arrives, it is added to the one of the books if it concerns a limit order (be it a buy or a sell order) for which no suitable counter-orders\(^6\) are available at that price. If it concerns a market order, or a limit order for which suitable counter-orders are available, then the transaction between the involved parties is carried out. Note that it can occur that a limit order is only partially carried out, if the number of securities in that order is larger than the number of available securities in the suitable counter-orders.

The PLAT project has spurred the creation of new flavors of trading strategies incorporating the limit-order book data which is seen as an expression of market sentiment and/or as a guide for order placement. There exist, however, a number of drawbacks in the PLAT simulator: i) the likely response of the external market to the actions taken by the trading agents is not simulated (a response which would become more likely as the volumes traded by the agents become larger and larger) and ii) no commission or tax charges are included in the simulation, and thus frequent trading is not penalized.

The yearly competitions held in the framework of the PLAT project proved invaluable for evaluating a large number of trading strategies, until the project

\(^6\)For a buy order, a suitable counter-order is a sell order at or below the price of the buy order. For a sell order, accordingly, a suitable counter-order is a buy order at or above the price of the sell order.
3.3. Trading agents in more realistic circumstances

was discontinued in 2006. The project’s website[7] still provides references to the most important research results.

In the second part of this dissertation, we want to investigate thoroughly the performance of a recent and fairly advanced trading strategy described in the literature. Based on this actual and first-hand experience, we wish to gain a clearer understanding of what is possible within the boundaries of today’s technology.

4.1 The adaptive reinforcement learning trading agent

The adaptive reinforcement learning (ARL) trading agent was introduced by Dempster and Leemans (2006). It is intended for trading on foreign-exchange markets and is essentially a three-layered structure consisting of i) a neural network, ii) a risk management overlay and iii) a dynamic utility optimization layer. This particular design is an attempt to resolve some of the common problems found in automated trading systems: i) failure to adapt to trend changes, resulting in a reduced profitability or even large drawdowns in profits during operation, and ii) excessive switching behaviour resulting in very high transaction costs. We will now discuss in some more detail the design of each
4.1. The adaptive reinforcement learning trading agent

layer. Thereby, we will also try to point to and sometimes address some of the mathematical ambiguities that are present in [Dempster and Leemans 2006].

- **Neural network layer**: The neural network is specified as a recurrent single layer neural network, which takes a series of \( n \) past returns \( r_j(t_i) = p(t_i-j) - p(t_i-j-1) \) \((j = 0...n - 1)\) at the input and outputs the position \( F(t_i) \in \{-1, 1\} \) to take, *out of or in* the market\(^1\), respectively:

\[
F(t_i) = \text{sign} \left( \sum_{j=0}^{n-1} w_j(t_i)r_j(t_i) + w_n(t_i)F(t_{i-1}) + v(t_i) \right),
\]

where \( w \) and \( v \) denote the weight vector and threshold of the neural network. Notice also the dependency on the previous output \( F(t_{i-1}) \), which makes the network recurrent.

If we assume that interest rates can be ignored, then the cumulative profit per unit of \( P(t_i) \) at a time \( t_i \) can be calculated as:

\[
P(t_i) = \sum_{j=0}^{i} R(t_j),
\]

with the returns \( R(t_j) \) defined as:

\[
R(t_j) = F(t_{j-1})r_0(t_j) - \frac{\delta}{2} |F(t_j) - F(t_{j-1})|,
\]

in which \( \delta \) denotes the transaction cost per trade, which equals the *bid-ask spread*, i.e. the difference between the ask and the bid price of one unit of the foreign currency\(^2\). Note that the above expression also considers the profits that can be made on a downstep of the foreign currency.

\(^1\)Note that with *out* we actually take a position with which we can generate profit on a price decrease of the foreign currency (e.g., a loan in the foreign currency).

\(^2\)In eg. stock markets, the transaction costs also include brokerage commissions and fees, which vary with respect to investor type (individuals, institutions or market makers) and trade size.
4.1. The adaptive reinforcement learning trading agent

The neural network described above will be trained to provide a maximal risk-adjusted return (rather than maximizing pure returns). The Sharpe ratio $S$ is a widely-used measure of risk-adjusted return, and is defined (in the absence of a risk-free return, see above) as:

$$S = \frac{\text{Average}(\{R(t_i)\})}{\text{Standard deviation}(\{R(t_i)\})}. \tag{4.4}$$

Generally, it holds that the higher the Sharpe ratio, the better. This can either be the result of a higher average return or either of a reduced volatility in the returns (hence making the average return more guaranteed).

In the ARL trading agent, a related objective function is employed, called the moving differential Sharpe ratio. It is obtained by considering moving estimates of the first ($A(t_i)$) and second ($B(t_i)$) moments of $\{R(t_i)\}$ expanded in an adaptation rate parameter $\eta$ around the previous values of these moments, $A(t_{i-1})$ and $B(t_{i-1})$, respectively:

$$A(t_i) = A(t_{i-1}) + \eta(R(t_i) - A(t_{i-1})),$$
$$B(t_i) = B(t_{i-1}) + \eta(R(t_i)^2 - B(t_{i-1})). \tag{4.6}$$

Using (4.6), the Sharpe ratio $S_\eta(t_i)$ at a time $t_i$ becomes:

$$S_\eta(t_i) = \frac{A(t_i) + \eta(R(t_i) - A(t_{i-1}))}{\sqrt{A(t_i) + \eta(R(t_i) - A(t_{i-1})) - (A(t_i) + \eta(R(t_i) - A(t_{i-1})))^2}}. \tag{4.7}$$

---

3 The $k$-th moment $P_k^X$ of a series of $n$ values $X_i$ ($i = 1...n$) is defined as:

$$P_k^X = \frac{1}{n} \sum_{i=1}^{n} X_i^k. \tag{4.5}$$
4.1. The adaptive reinforcement learning trading agent

Expanding (4.7) to first order in the adaptation rate $\eta$, we obtain:

$$S_\eta(t_i) \approx S_\eta(t_i)|_{\eta=0} + \eta \left. \frac{dS_\eta(t_i)}{d\eta} \right|_{\eta=0} + O(\eta^2)$$

$$= S(t_{i-1}) + \eta \left. \frac{dS_\eta(t_i)}{d\eta} \right|_{\eta=0} + O(\eta^2). \quad (4.8)$$

From (4.8) it becomes clear that the zeroth order term in $S_\eta(t_i)$ is already a fixed value at a time $t_i$, and only the first order term is still a candidate for optimization. Therefore we define the moving differential Sharpe ratio $D(t_i)$ as:

$$D(t_i) = \left. \frac{dS_\eta(t_i)}{d\eta} \right|_{\eta=0}$$

$$= \frac{B(t_{i-1})(R(t_i) - A(t_{i-1})) - \frac{1}{2}A(t_{i-1})(R(t_i)^2 - B(t_{i-1}))}{(B(t_{i-1}) - A(t_{i-1})^2)^{3/2}}. \quad (4.9)$$

Notice that the only new information required to update $D(t_i)$ at a time $t_i$ is the knowledge of $R(t_i)$.

Now that we have defined our objective function as $D(t_i)$, we can train our neural network to perform well in that respect. This is done by adjusting the weight vector with a simple gradient ascent optimization algorithm:

$$w_j(t_i) = w_j(t_{i-1}) + \rho \Delta w_j(t_i) \quad j = 0...n, \quad (4.10)$$

where $\rho$ denotes a learning rate parameter. $\Delta w_j(t_i)$ is defined as:

---

4 Naturally, more advanced optimization algorithms can be employed here, too.
4.1. The adaptive reinforcement learning trading agent

\[ \Delta w_j(t_i) = \frac{dD(t_i)}{dw_j(t_i)} \]

\[ = \sum_{k=0}^{i} \frac{dD(t_i)}{dR(t_k)} \left\{ \frac{dR(t_k) dF(t_k)}{dF(t_i)} \frac{dF(t_k)}{dR(t_i)} + \frac{dR(t_k) dF(t_{k-1})}{dF(t_i-1)} \frac{dF(t_{k-1})}{dR(t_i)} \right\} \] (4.11)

The above expression is approximated for an on-line update by considering only the term that depends on the most recent return \( R(t_i) \) and by assuming that \( \frac{dF(t_{i-1})}{dw_j(t_i)} \approx \frac{dF(t_{i-1})}{dw_j(t_{i-1})} \):

\[ \Delta w_j(t_i) \approx \frac{dD(t_i)}{dR(t_i)} \left\{ \frac{dR(t_i) dF(t_i)}{dF(t_i)} \frac{dF(t_i)}{dR(t_i)} + \frac{dR(t_i) dF(t_{i-1})}{dF(t_i-1)} \frac{dF(t_i-1)}{dR(t_i)} \right\} \]. (4.12)

Note that in the above elaboration we have assumed the differentiability of \( \{ F(t_i) \} \) and \( \{ R(t_i) \} \), while in fact they are defined as discrete and incontinuous, respectively. This can be resolved by approximating – during the optimization of the neural network only – the \( sign \) function by a \( \tanh \) function.\[ ^5 \]

The different components of the above equation simplify to:

\[ \frac{dD(t_i)}{dR(t_i)} = \frac{B(t_{i-1}) - A(t_{i-1})R(t_i)}{(B(t_{i-1}) - A(t_{i-1})^2)^{3/2}}, \]

\[ \frac{dR(t_i)}{dF(t_i)} = -\frac{\delta}{2} \tanh(F(t_i) - F(t_{i-1})), \]

\[ \frac{dF(t_i)}{dw_j(t_i)} \approx \frac{\partial F(t_i)}{\partial w_j(t_i)} + \frac{\partial F(t_i)}{\partial F(t_{i-1})} \frac{dF(t_{i-1})}{dw_j(t_{i-1})} \]

\[ = (1 - F(t_i)^2) (r_0(t_i) + w_n(t_i) \frac{dF(t_{i-1})}{dw_j(t_{i-1})}), \]

\[ \frac{dR(t_i)}{dF(t_i)} = r(t_i) + \frac{\delta}{2} \tanh(F(t_i) - F(t_{i-1})). \] (4.13)

\[ ^5 \frac{d\text{abs}(x)}{dx} = \text{sign}(x) \text{ for } x \neq 0. \]
A similar discussion can be held for the threshold $v(t_i)$ of the neural network, which is then optimized accordingly through:

$$v(t_i) = v(t_{i-1}) + \rho \Delta v(t_i),$$

$$\Delta v(t_i) \approx dD(t_i) \left( dR(t_i) \frac{dF(t_i)}{dv(t_i)} + \frac{dR(t_i)}{dF(t_{i-1})} \frac{dF(t_{i-1})}{dv(t_{i-1})} \right),$$

$$\frac{dF(t_i)}{dv(t_i)} \approx \frac{\partial F(t_i)}{\partial v(t_i)} + \frac{\partial F(t_i)}{\partial F(t_{i-1})} \frac{dF(t_{i-1})}{dv(t_{i-1})}$$

$$= (1 - F(t_i)^2)(1 + w(t_i) \frac{dF(t_{i-1})}{dv(t_{i-1})}).$$

Initially, $\{w(t_i)\}$ and $v(t_i)$ are chosen from a normal distribution (mean = 0.0, standard deviation = 1.0). If the sum of the absolute values of $\{w(t_i)\}$ and $v(t_i)$ exceed a specified threshold value, they are automatically scaled back by $a (0 < a < 1)$ for an indefinite number of times, until they no longer do so.

- **Risk management overlay**: The risk management overlay attempts to limit the drawdown in the case where the trading agent stops being profitable. If at a certain point the drawdown in the cumulative profit becomes greater than a parameter $z$, the trading agent is automatically shut down (for re-optimization or re-design by its human supervisor). This layer also manages a trailing stop-loss for every trade. A stop-loss is set and adjusted so that it is always a parameter $x$ under or above the best price ever reached during the life of the position. If a position is closed out before a trade exit signal was given, this behaviour was clearly not expected by the underlying neural network layer. Hence, the market behaves contrary to the model, and it is likely that this market behaviour will persist for some time. Therefore, a freeze-down interval
4.1. The adaptive reinforcement learning trading agent

$u$ is imposed\footnote{As the occurrence of these cool-down periods should be minimal, $u$ is chosen as a fixed parameter, which will not be subject to optimization by the uppermost layer.} during which the system is forced to assume the same trading position. A final feature of the risk management layer involves the evaluation of the strongness of the trading signal $F(t_i)$. A trading signal is only effectively carried out if the absolute value of the signal in \(F(t_i)\) before discretization (i.e., before taking the \textit{sign} function) is greater than a threshold parameter $y$. This effectively prevents most of the unprofitable oscillatory switching behaviour.

- **Dynamic utility optimization layer**: The uppermost layer of the ARL trading agent is in charge of the dynamic optimization of the parameters of the ARL (i.e. $\delta, \eta, \rho, x,$ and $y$)\footnote{Note that we will, in the spirit of Dempster and Leemans (2006), treat the transaction cost per trade $\delta$ as an optimization parameter, i.e. we want to know the value of $\delta$ for an optimized ARL trading agent. Moreover, the market data provided by Dukascopy does not even contain the spreads, needed for the calculation of the transaction costs.} which reflect the changes needed to allow the underlying two layers to adapt to changing market conditions (rather than price changes, which is essentially the task of the neural network). For example, it may be very well possible that in times of higher volatility, a larger volatility parameter is needed to allow the neural network to react to rapid price changes. We define the $n$-interval utility function $U_n$, which constitutes a trade-off between maximizing the (average) return and minimizing the number of negative returns, effectuated in the difference between two indicators $\bar{R}_n$ and $\Sigma_n$, respectively:

$$U_n(R, \Sigma, \nu, \delta, \eta, \rho, x, y) = a(1 - \nu)\bar{R}_n(\delta, \eta, \rho, x, y) - \nu\Sigma_n(\delta, \eta, \rho, x, y),$$

(4.15)
where:

\[
\Sigma_n = \frac{\sum_{i=1}^{n} R(t_i)^2 (R(t_i) < 0)}{\sum_{i=1}^{n} R(t_i)^2 (R(t_i) > 0)},
\]

\[
\bar{R}_n = \frac{\sum_{i=1}^{n} R(t_i)}{n},
\] (4.16)

and \(\nu\) \((0 < \nu < 1)\) indicates a risk-aversion parameter (a higher \(\nu\) means a higher risk aversion), and \(a\) a predefined (fixed) parameter which is intended to fix the natural imbalance between the two indicators. The numerator of \(\Sigma_n\) contains only negative returns, the denominator only positive returns. Note that \(n\) denotes a set of out-of-sample data points obtained in operation (i.e. with an optimized neural network).

The utility function defined in (4.15) is, similar as for the weight vector and threshold of the neural network, subject to optimization. Since, however, this function depends equally on all the previously generated returns, the calculation of the gradients with respect to \(\delta, \eta, \rho, x,\) and \(y\) comes at a high computational cost, which essentially increases linearly with \(n\). Therefore, we prefer to search over the 5-dimensional parameter space directly, rather than having to calculate the gradients (and as an added benefit, we also avoid being stuck in local maxima).

The direct-search algorithm is organized as follows:

1. We define (in an arbitrary way) the reasonable boundaries in which a valid set of the five parameters should be confined.

2. Each parameter is assigned a random value within these boundaries.

3. We optimize consecutively in each of the five parameters separately,
4.2 Some remarks on the implementation

by evaluating a number of random values distributed triangularly around the current best parameter value and adopting a new best parameter value if it exceeds the current best value for the utility function (4.15).

4. Step 3 is repeated until the best value for the utility function (nearly) doesn’t exceed any more after one optimization round over all five parameters (= convergence).

Of course, this scheme assumes a limited interdependency between the parameters of the utility function.

4.2 Some remarks on the implementation

The ARL trading agent was programmed in the Python programming language. Because the code writing is an integral part of this master dissertation, and also for reference purposes, we chose to reproduce the full source code in Appendix A. Python is an interpreted language, and thus does not require compilation. The code requires an additional library to be installed, called Numpy. The ARL trading agent is fully command-line driven, and outputs

---

$\text{The triangular distribution } f_{\text{triang}} \text{ is a continuous probability distribution with lower limit } a, \text{ mode } c, \text{ and upper limit } b:$

$$f_{\text{triang}}(x|a,b,c) = \begin{cases} \frac{2(x-a)}{(b-a)(c-a)} & a \leq x \leq c \\ \frac{2(b-x)}{(b-a)(b-c)} & c \leq x \leq b \\ 0 & \text{otherwise} \end{cases}$$

---

9http://www.python.org/
10http://numpy.scipy.org/
4.3 Priliminary testing of the ARL trading agent

status messages during operation, as illustrated in Figure 4.1. The syntax to start the agent is as follows:

```python
python arl.py number_of_weights risk_aversion prices.txt output.txt
```

where:

- `number_of_weights` = integer number, defining the number of components in the weight vector
- `risk_aversion` = floating number between 0 and 1, defining the risk aversion parameter ’nu’
- `prices.txt` = text file from which prices are read, one price on each line
- `output.txt` = text file into which the generated returns are written

4.3 Priliminary testing of the ARL trading agent

In this section we will perform some preliminary tests of the ARL trading agent. We will verify that the agent is functioning as expected (and at the same time gain more insight in some of its peculiarities) by conditioning the agent to the prediction of a periodically repeated normal distribution (mean = 0.0, standard deviation = 1.0) of returns. The returns are expressed in units of pip[^11] and were rounded up to the nearest round number. The primitive data set, shown in Figure 4.2 contains 16 returns, this data set is repeated continuously.

[^11]: A pip is the smallest unit by which a currency may change value.
4.3. Preliminary testing of the ARL trading agent

Figure 4.1: The ARL trading agent in action.

```bash
# Python code snippet
```
4.3 Preliminary testing of the ARL trading agent

Figure 4.2: Illustration of the primitive data set, consisting of 16 returns (expressed in units of pips).
4.3. Preliminary testing of the ARL trading agent

The data signal being periodic, this should in principle be a fairly easy task for our trading agent. Assuming no transacting costs ($\delta = 0$), it can be inferred from Figure 4.2 that on average one pip per time step can be made theoretically. A first test constitutes of varying the number of components in the weight vector, and here we should normally see the trading agent getting closer to the theoretical maximal percentage gain (i.e. when the maximal return is made in every out-of-sample interval, and assuming no transaction costs) as the ARL gains flexibility. Note that we actually optimize with respect to the moving differential Sharpe ratio, i.e. a risk-adjusted return indicator. Therefore, the maximal attainable gain is not an optimization goal in se, but nonetheless it is often a major way to accomplish the optimization of the moving differential Sharpe ratio. The optimization procedure carried out by the uppermost layer is not yet submitted to a test here, as fixed (and considered reasonable) values have been chosen: $\delta = 0.0$, $\eta = 0.2$, $\rho = 0.05$, $x = 100.0$, and $y = 0.0$.

In Figure 4.3, we observe the quite steady improvement of the mean percentage gains with increasing number of components in the weight vector, which is a satisfying result. It is important to note, however, that the weight vector and the threshold are initialized by a random process, and that as a result the gradient ascent optimizer gets stuck quite often in a nearby local maximum, an effect that is even more likely to occur as the number of dimensions (in which an optimal point is searched) increases. This is why repetitive runs using the same number of components in the weight vector can result in slightly different gains. Figure 4.3 shows the average result for 10 consecutive runs, and error bars denote the standard deviation from the mean. Ideally, we would like both an increase of the average gain and a decrease of the standard deviation through optimization of the set of parameters $\delta$, $\eta$, $\rho$, $x$, and $y$ for this particular

\footnote{Unfortunately, Dempster and Leemans (2006) shed no light on how they handled this quite relevant issue.} 
4.3. Preliminary testing of the ARL trading agent

Figure 4.3: Mean percentage of the theoretical maximal gain for 10 consecutive runs, with respect to the number of weights $w$. Error bars denote the standard deviation from the mean. $\delta = 0.0$, $\eta = 0.2$, $\rho = 0.05$, $x = 100.0$, and $y = 0.0$. Out-of-sample data points $= 4000$ (hence, the theoretical maximal gain equals 4000 pips), re-training of the neural network every 500 data points, using 2000 data points of past returns. The number of training epochs was 10.
4.3 Preliminary testing of the ARL trading agent

Table 4.1: Boundaries of the five-dimensional parameter space.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Minimum</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\delta$</td>
<td>0.0</td>
<td>1.0</td>
</tr>
<tr>
<td>$\eta$</td>
<td>0.0</td>
<td>1.0</td>
</tr>
<tr>
<td>$\rho$</td>
<td>0.0</td>
<td>1.0</td>
</tr>
<tr>
<td>$x$</td>
<td>0</td>
<td>200</td>
</tr>
<tr>
<td>$y$</td>
<td>0.0</td>
<td>3.0</td>
</tr>
</tbody>
</table>

data stream (see below).

A next step consists of testing the optimization scheme present in the uppermost layer of the ARL trading agent. Adopting parameter values of $\nu = 0.5$ and $a = 1.0$, the maximal theoretical value of the utility function (4.15) for 4000 out-of-sample data points is 0.5. Figure 4.3 shows that through optimization within arbitrarily defined boundaries of the five parameters (see Table 4.1) this value is reached almost exactly from $w = 6$ and above after only two iterations, again a satisfying result. For $w < 6$ the initial improvements of the utility function are quite significant, but it appears that the limited flexibility inherent to such a small weight set prevents further improvement from being made. Analysis of the five parameters during the optimization reveals the changes in the adaptation rate parameter $\eta$, the stop-loss parameter $x$, and the threshold parameter $y$ as the primary factors in achieving a higher value for the utility function. The transaction cost $\delta$ and the learning rate parameter remain virtually constant.
4.3. Preliminary testing of the ARL trading agent

Figure 4.4: Best value for the utility function $U_{4000}$ \((4.15)\) in five consecutive iterations of optimizations in all five parameters with respect to the number of weights $w$. Starting parameters are $\delta = 0.0$, $\eta = 0.2$, $\rho = 0.05$, $x = 100.0$, and $y = 0.0$. Out-of-sample data points = 4000, re-training of the neural network every 500 data points, using 2000 data points of past returns. The number of training epochs was 10. The red line depicts the maximal theoretical value of the utility function for 4000 out-of-sample data points.
4.4 The ARL trading agent put to the real test

Now that we have acquired already some hands-on experience in handling the ARL trading agent, it is time to put it to a real test: using real past foreign exchange data, we want to learn if the trading agent can be more profitable than a simple buy-and-hold strategy.

4.4.1 The data

The data used in this thesis work is the 10-minute interval FX rate of the United States dollar against the European Union euro from October the first of 2006 until December the 31th of 2007. This data was obtained from the Dukascopy Swiss Forex Group, a company which provides a free, comma-separated-values (CSV) data export. The foreign exchange market is open 24 hours a day, and is closed only during weekends, and therefore the constant prices from weekends were removed from the data set. Apart from that, no further preprocessing was carried out. The first 10000 data points (corresponding roughly with three months of data) are used for initial optimization of the ARL trading agent.

The Euro-dollar exchange rate is reproduced in Figure 4.5 and shows that we will face some tough competition from a buy-and-hold strategy, which would have generated a profit (omitting transaction costs) of approximately 11.4% in the data-point interval 10001-46872 (which corresponds roughly with the year 2007).

---

13 This data is included as supplemental information in the electronic version of this work.
14 http://www.dukascopy.com/
15 This data can be used for informational purposes only.
4.4. The ARL trading agent put to the real test

Figure 4.5: The Euro-dollar exchange rate (01/10/06 - 31/12/07). The source data for this graph was collected from Dukascopy’s CSV data export.
4.4. The ARL trading agent put to the real test

4.4.2 The results

Three different numbers of weights (8, 16, and 32, respectively, corresponding with 80, 160, and 320 minutes of past data) were used in the simulations. All of them represent relatively short periods of time, and thus force the focus of the ARL trading agent upon short-term trends, as long-term trends are kept more or less invisible to the agent. The results of the three runs are shown in the upper half of Table 4.2. It turns out that the ARL trading agent in its current implementation is extremely profitable, with generated percentage profits ranging from 55.0% to 72.2% during the simulation period (which corresponds roughly with one year), the worst result (originating from the configuration using 32 weights) probably being caused by the optimizer getting stuck in local maxima quite often. The gains are also made quite consistent over time, as illustrated in Figure 4.6. The trading agent apparently exploits many of the short-term trends, and outperforms the buy-and-hold strategy for all numbers of weights by a considerable margin. However, one should not forget that we explicitly allowed the optimization of the transaction cost $\delta$, which always returned $\delta = 0$.\[16\] We have redone the simulation using 16 weights (i.e. the one that produced the best result, see the upper half of Table 4.2), but now by constraining the transaction cost to fixed values of 0.5, 1, 1.5, and 2 pips, respectively. From the lower half of Table 4.2 we observe a sharp decrease in profitability (despite the significant lower number of trades as compared to the $\delta = 0$ case), with the ARL trading agent being outperformed by the buy-and-hold strategy as soon as the transaction cost is elevated to 0.5 pip, and with the agent becoming (completely) unprofitable at the 1 pip mark. In fact, for these

\[16\] This looks quite obvious, but nonetheless a higher transaction cost has a limiting effect on the switching behaviour of the trading agent, and may actually help in finding a more profitable state of the agent during the optimization of the five parameters.
4.4. The ARL trading agent put to the real test

simulations (alone), we had to disable the risk management overlay to prevent the ARL trading agent from exiting prematurely.

Before drawing any definite conclusions, we first want to get some insight in the spread of the profits that are caused by the random process with which the weight vector and the threshold are (re-)initialized at several points during the simulation, by performing multiple simulations on both a constraint-free ($\delta = 0$) and on a constrained ($\delta = 0.5$) 16-weights ARL trading agent. The number of simulations in each case was restricted to 6 only, because of the quite severe associated computational cost.\footnote{The computational cost for one simulation with 16 weights applied to the data of Figure 4.5 is about 36 hours on (one core of) an Intel Xeon 5130 CPU.} Although the standard deviation calculated from these simulations will therefore probably not be really tightly converged, it does show some variation in the results (standard deviations of 22.5\% and 9.0\%, respectively), as can be seen in Table 4.3. This uncertainty is an additional risk factor to take into account in the current implementation of the trading agent. Moreover, these simulations also reveal that the average percentage gain from the constrained simulations ($\delta = 0.5$) is actually slightly negative. However, consecutive constraint-free simulations all easily outperformed the buy-and-hold strategy, which strengthens our belief that these large profits were not produced by coincidence, but rather follow from genuine trends in the data points that were picked up by the trading agent.

An important factor in the final evaluation of the performance of the ARL trading agent concerns of course the real transaction cost. Goodhart et al. (2002) computed the average spread in the euro-dollar exchange rate from September 28th 1999 to March 8th 2000, and found them to be well above the 2-pips level. However, McGroarty et al. (2006) employed a slightly different calculation method on the same data set, and situated the bid-ask spreads
4.4. The ARL trading agent put to the real test

<table>
<thead>
<tr>
<th>Weights</th>
<th>Pips gained</th>
<th>Percentage gain</th>
<th># trades</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>9452</td>
<td>72.2</td>
<td>nc</td>
</tr>
<tr>
<td>16</td>
<td>10088</td>
<td>77.0</td>
<td>17199</td>
</tr>
<tr>
<td>32</td>
<td>7206</td>
<td>55.0</td>
<td>nc</td>
</tr>
<tr>
<td>16 (δ = 0.5 pips)</td>
<td>461</td>
<td>3.5</td>
<td>15668</td>
</tr>
<tr>
<td>16 (δ = 1.0 pips)</td>
<td>−599</td>
<td>−4.6</td>
<td>994</td>
</tr>
<tr>
<td>16 (δ = 1.5 pips)</td>
<td>−3133</td>
<td>−23.0</td>
<td>2400</td>
</tr>
<tr>
<td>16 (δ = 2.0 pips)</td>
<td>−1817</td>
<td>−13.9</td>
<td>1015</td>
</tr>
</tbody>
</table>

mostly between 0.5 and 1.0 pip. In either case, the trading agent would likely not be able to generate consistent profits when real transaction costs are used. This, however, does not necessarily mean that no alterations can be thought of which would result in greater profitability. One obvious possibility would be to focus more on longer-term trends when high transaction costs are involved (something which can be done by considering a larger past time window), thereby further reducing the number of trades that are carried out. Apart from that, we could also envisage implementing a more advanced (and probably also computationally more expensive) optimization algorithm for the neural network layer (mainly to overcome the problem of getting stuck in local maxima).
4.4. The ARL trading agent put to the real test

Figure 4.6: Cumulative returns as a function of time for three different numbers of weights. The details of this simulation have already been described in Table 4.2. Note that the first 10000 points are used for initial optimization of the ARL trading agent, therefore no profits are generated there. For visual reference only (since the scale on the $y$-axis is incorrect), we have reprinted (in light-gray) the euro-dollar exchange rates from Figure 4.5.
### 4.4. The ARL trading agent put to the real test

Table 4.3: Percentage gain and percentage deviation for 6 simulations of both a constraint-free ($\delta = 0$) and on a constrained ($\delta = 0.5$) 16-weights ARL trading agent using the euro-dollar exchange rate data of Figure 4.5. Other starting parameters are $\eta = 0.2$, $\rho = 0.05$, $x = 100.0$, and $y = 0.0$. Re-training of the neural network every 500 data points for 10 training epochs, using 2000 data points of past returns. (Re-)optimization of all five (four if $\delta$ is constrained) parameters every 10000 data points for five iterations. Optimizations were performed in the parameter space defined in Table 4.1. nc = not counted.

<table>
<thead>
<tr>
<th>Transaction cost</th>
<th>Percentage gain</th>
<th>Percentage standard deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\delta = 0$ pips</td>
<td>59.4</td>
<td>22.5</td>
</tr>
<tr>
<td>$\delta = 0.5$ pips</td>
<td>$-0.2$</td>
<td>9.0</td>
</tr>
</tbody>
</table>
4.4. The ARL trading agent put to the real test
In this dissertation, we have attempted to gain insight in the profitability of technical trading strategies, both by consulting the available scientific literature on the topic, and by studying a fairly recent advanced trading agent.

From our literature review, we learnt that many of the earlier studies lacked the proper testing procedures to provide conclusive evidence on the (un-)profitability of technical trading. Some of the frequently occurring errors included the consideration of too few strategies, the absence of statistical significance tests, no consideration of the risk-return trade-off, the awkward presentation of results, and data snooping bias. Modern studies have generally improved upon the aforementioned shortcomings of earlier studies, and provide (not yet conclusive) evidence that the efficient market hypothesis may not hold in all markets and/or at all times, hence supporting the concept of technical trading. The prevailing theoretical models included noise in equilibrium prices, behaviour models, herd behaviour, market power, and chaos. Among the empirical explanations we distinguished intervention of central banks, order flow, temporary market inefficiencies, and risk premiums.

The second and most important part of this dissertation involved the study
Conclusions

of the adaptive reinforcement learning trading agent introduced by [Dempster and Leemans (2006)]. Applied to historical euro-dollar foreign exchange rates, we observed that if the transaction costs (which equal to the bid-ask spreads) are not taken into account, the ARL trading agent can be extremely profitable, with generated percentage profits ranging from 55.0% to 72.2% during the simulation period (which corresponds roughly with one year). Despite the fact that the weight vector and the threshold are initialized by a random process, multiple runs of the trading agent without transaction costs all easily outperformed the buy-and-hold strategy, thereby providing convincing evidence that these profits were not produced by coincidence, but rather follow from genuine trends in the data points that were picked up by the trading agent.

However, from simulations in which the transaction costs were constrained to non-zero values, we learnt that the trading agent would likely not be able to generate significant profits when real transaction costs are used (which are estimated to be at least 0.5 pip). This, however, does not necessarily mean that no alterations can be thought of which would result in greater profitability. One obvious possibility would be to focus more on longer-term trends, thereby further reducing the number of trades that are carried out. Apart from that, several optimizations to the trading agent itself can be conceived. Further research should concentrate on both of these issues.
Source code of the ARL trading agent

# Definitions
# debug functions and routines debug
# past returns glob_m
# sum of current and past values glob_nv
# checks function consistency debug
# hardness of signum approximation glob_hd
# profit glob_pft
# maximal profit glob_pftm
# break signal encountered glob_brk
# weights threshold glob_wthr
# training ticks glob_train_t
# training epochs glob_train_e
# test ticks glob_test
# index in data array glob_i
# number of training glob_j
# number of optimizations glob_k
# risk aversion glob_nu
# utility factor glob_utilityfactor
# small number definition glob_epsilon
# min/max glob_delta_down
# min/max glob_delta_up
# min/max glob_eta_down
# min/max glob_eta_up
# min/max glob_rho_down
Source code of the ARL trading agent

```python
# imports
import numpy
import sys

# class and function definitions

class Container:
    
    "Container of all objects/arrays/..."

    def __init__(self, m, nv, hd, train_t, train_e, test, opt_t, full):
```
#glob's
self.m = m
self.nv = nv
self.hd = hd
self.pft = 0.0
self.pftm = 0.0
self.brk = False
self.wthr = (float(self.m) + 1.0) * 5.0
self.train_t = train_t
self.train_e = train_e
self.opt_t = opt_t
self.test = test
self.i = 0
self.j = 0
self.k = 0
self.scale = 0.5
self.u = 0.0
self.utime = 10.0
self.nu = 0.5
self.utilityfactor = 1.0
self.epsilon = 0.000001
self.delta_down = 0.0
self.delta_up = 1.0
self.eta_down = 0.0
self.eta_up = 1.0
self.rho_down = 0.0
self.rho_up = 1.0
self.x_down = 0.0
self.x_up = 200.0
self.y_down = 0.0
self.y_up = 3.0
self.spoints = 10
self.smaxiter = 5

##l1's
self.w = numpy.zeros((self.nv,self.m + 1), float)
for i in range(0, self.nv):
    self.w[i] = numpy.random.standard_normal(self.m + 1)
self.f = numpy.zeros(self.nv, float)
self.df = numpy.zeros(self.nv, float)
self.v = numpy.zeros(self.nv, float)
selrt = numpy.zeros(1, float)
sel.at = numpy.zeros(self.nv, float)
sel.bt = numpy.zeros(self.nv, float)
sel.at[0] = 0.001
sel.bt[0] = 0.01
self.dftdwit = numpy.zeros((self.nv,self.m + 1), float)
sel.dftdvt = numpy.zeros(self.nv, float)

##l2's
self.x = numpy.zeros(1, float)
sel.y = numpy.zeros(1, float)
def shift(a):
    n = len(a)
    if (len(a.shape) == 1):
        return numpy.hstack((a[-1:], a[:-1]))
    elif (len(a.shape) == 2):
        return numpy.vstack((a[-1:], a[:-1]))
    else:
        print "error in dim in function shift"

def shiftfl(a, length):
    n = len(a)
    if (length % n != 0):
        if (len(a.shape) == 1):
            #return numpy.hstack((a[-length:], a[:n-length]))
            return numpy.hstack((a[-(n-length):], a[:length]))
        else:
            print "error in dim in function shiftfl"
    else:
        return a

def tanhf(x, hd, n):
    if (n == 0):
        return numpy.tanh(hd * x)
    elif (n == 1):
        return hd * (1.0 - numpy.tanh(hd * x)**2.0)
    else:
        print "error differiator n in function tanhf"

def layer1(C):
    "ARL layer!"
    C.f = shift(C.f)
    C.df = shift(C.df)
    C.at = shift(C.at)
    C.bt = shift(C.bt)
    C.dftdwit = shift(C.dftdwit)
    C.dftdvt = shift(C.dftdvt)
#calculation of f and df
sumf = 0.0
for i in range(0, C.m):
    sumf += C.w[0][i] * C.r[i]
    sumf += C.w[0][C.m] * C.f[1] + C.v[0]
C.f[0] = numpy.sign(sumf)
C.df[0] = tanhf(sumf, C.hd, 1)

#calculation of rt
C.rt[0] = C.f[1] * C.r[0] - C.delta[0] * numpy.abs(C.f[0] - C.f[1]) / 2.0

#calculation of At and Bt

#calculation of dD_t/dR_t

#calculation of dR_t/dF_t, dR_t/dF_t-1
C.drtdft = - C.delta[0] / 2.0 * tanhf(C.f[0] - C.f[1], C.hd, 0)
C.drtdftm1 = C.r[0] + C.delta[0] / 2.0 * tanhf(C.f[0] - C.f[1], C.hd, 0)

#calculation of dF_t/dw_it (and dF_t/dv_t)
for i in range(0, C.m):
    C.dftdwit[0][i] = C.df[0] * ( C.r[i] + C.w[0][C.m] * C.dftdwit[1][i] )
    C.dftdwit[0][C.m] = C.df[0] * ( C.f[1] + C.w[0][C.m] * C.dftdwit[1][C.m] )
    C.dftdvt[0] = C.df[0] * ( 1.0 + C.w[0][C.m] * C.dftdvt[1] )

#calculation of w_it and v_t
C.w = shift(C.w)
for i in range(0, C.m):
    C.w[0][i] = C.w[1][i] + C.rho[0] * C.ddtdrt * ( C.drtdft * C.dftdwit[0][i] + C.drtdftm1 * C.dftdwit[1][i] )
    C.w[0][C.m] = C.w[1][C.m] + C.rho[0] * C.ddtdrt * ( C.drtdft * C.dftdwit[0][C.m] + C.drtdftm1 * C.dftdwit[1][C.m] )

C.v = shift(C.v)

#calculation of final f using the new weights
sumf = 0.0
for i in range(0, C.m):
    sumf += C.w[0][i] * C.r[i]
    sumf += C.w[0][C.m] * C.f[1] + C.v[0]
C.f[0] = numpy.sign(sumf)

#check for strongness of signal (l2, but works best here)
if ((C.f[0] != C.f[1]) and (C.f[1] != 0.0)):
    if (numpy.abs(sumf) <= C.y[0]):  
        C.f[0] = C.f[1]

#calculation of correct rt as presented to the upper layer
Source code of the ARL trading agent

```python
C.rt[0] = C.f[1] * C.r[0] - C.delta[0] * numpy.abs(C.f[0]-C.f[1]) / 2.0

#weights management
while (numpy.sum(numpy.abs(C.w[0])) + numpy.abs(C.v[0]) > C.wthr):
    C.w *= C.scale
    C.v *= C.scale
return C

def layer2(C):
    "ARL layer2"
    if (C.u == 0):
        C = layer1(C)
    else:
        C.u -= 1

    #stop loss management
    if (C.f[0] != C.f[1]):
        C.xcum[0] = 0.0
    else:
        C.xcum[0] += C.r[0]
        if (C.f[0] == 1.0):
            if (C.xcum[0] < -C.x[0]):
                C.f[0] = -1.0
                C.rt[0] = C.f[1] * C.r[0] - C.delta[0] * numpy.abs(C.f[0]-C.f[1]) / 2.0
                C.xcum[0] = 0.0
                C.u = C.utime
        if (C.f[0] == -1.0):
            if (C.xcum[0] > C.x[0]):
                C.f[0] = 1.0
                C.rt[0] = C.f[1] * C.r[0] - C.delta[0] * numpy.abs(C.f[0]-C.f[1]) / 2.0
                C.xcum[0] = 0.0
                C.u = C.utime

    #calculation of the cumulative profit
    C.pft += C.rt[0]
    #maximum draw-down check (*must* be after stop loss management!)
    if (C.pft > C.pftm):
        C.pftm = C.pft
    if (C.pft < C.pftm - float(C.z[0])):
        #print "Error :: C.pft < C.pftm - C.z[0]."
        C.brk = True

    return C

def layer3(C, main):
```

58
if (C.opt_t < C.train_t):
    print "Error :: opt_t < train_t."
    return

if (C.i == 0 and main):
    print ':: Initializing ARL on new data string ::'
    print '% s % i % s' % (':: First ', C.train_t, \
    'data points (=train_t) are only used to train the arl. ::')

#Updating array of returns
C.rtrain[C.i - C.j * C.test] = C.rta[0]
if (main):
    C.ropt[C.i - C.k * C.opt_t] = C.rta[0]
C.r[0] = C.rta[0]

if (C.i - C.j * C.test + 1 == C.train_t):
    if (C.i >= C.train_t):
        print '% s % i % s % 7.3f % s' % (':: Cumulated profit at', \
            C.i + 1, 'data points :', C.pft, '::')
        #print '% s % i % s % i % s ' % (':: Received', C.i + 1, \
        'data points, (RE)TRAINING nn for', C.train_e, 'cycles ::')

#temp Container
D = Container(C.m, C.nv, C.hd, C.train_t, C.train_e, C.test, C.opt_t, \
    False)
D.x = numpy.copy(C.x)
D.y = numpy.copy(C.y)
D.z = numpy.copy(C.z)
D.delta = numpy.copy(C.delta)
D.eta = numpy.copy(C.eta)
D.rho = numpy.copy(C.rho)

#train temp Container
for m in range(0, C.train_e):
    lsp = 0
    lsm = 0
    lse = 0
    D.pft = 0.0
    D.pftm = 0.0
    for n in range(0, C.train_t):
        D.r = shift(D.r)
        D.r[0] = C.rtrain[n]
        D = layer2(D)
        if (m+1 == C.train_e):
            if (D.f[0] == 1.0):
                lsp += 1
    59
elif (D.f[0] == -1.0):
    lsm += 1
else:
    lse += 1

# if (main):
#     print "% s % i % s % 7.3f % s" % 
#         (": Training cycle", m + 1, "profit: ", D.pft, ":")
# else:
#     print "% s % i % s % 7.3f % s" % 
#         (": Training cycle", m + 1, 
#          "profit: ", D.pft, ":")

if (m+1 == C.train_e):
    print "% s % i % s" % ("statistics: ", lsp, lsm, lse)
    print D.w[0], D.v[0]

# store important values in container C
C.at = numpy.copy(D.at)
C.bt = numpy.copy(D.bt)
C.f = numpy.copy(D.f)
C.df = numpy.copy(D.df)
C.dftdvt = numpy.copy(D.dftdvt)
C.dftdwit = numpy.copy(D.dftdwit)
C.v = numpy.copy(D.v)
C.w = numpy.copy(D.w)

# Counter j + 1
C.j = C.j + 1

# shift C.rtrain by C.test places
shiftfl(C.rtrain, C.test)

if ((C.i - C.k * C.opt_t + 1 == C.opt_t) and main):
    print "% s % i % s" % (":: Received ", C.i + 1, 
"data points, (RE)OPTIMIZING arl ::")

siter = 0
while (siter < C.smaxiter):
    for counter in range(-1, 6):
        x_max = C.x[0]
y_max = C.y[0]
delta_max = C.delta[0]
eta_max = C.eta[0]
rho_max = C.rho[0]

        # range = -1 -> calculate current E:
        if (counter == 0):
            60
searchlist = numpy.random.triangular(C.x_down, x_max, C.x_up, \nC.spoints)
elif (counter == 1):
    searchlist = numpy.random.triangular(C.y_down, y_max, C.y_up, \nC.spoints)
elif (counter == 2):
    searchlist = numpy.random.triangular(C.delta_down, delta_max, \nC.delta_up, C.spoints)
elif (counter == 3):
    searchlist = numpy.random.triangular(C.eta_down, eta_max, \nC.eta_up, C.spoints)
elif (counter == 4):
    searchlist = numpy.random.triangular(C.rho_down, rho_max, \nC.rho_up, C.spoints)
elif (counter == -1 and siter == 0):
    print ":: Evaluating utility function at original position \nof the parameters ::"
elif (counter == 5 and siter + 1 == C.smaxiter):
    print ":: Evaluating utility function at final position \nof the parameters ::"
else:
    print "Nothing to do.", counter, siter, C.smaxiter
spoints = C.spoints
if (counter == -1):
    spoints = 1
# print "xxx", counter, siter, C.smaxiter, spoints
for n in range(0, spoints):
    # temp Container
    E = Container(C.m, C.nv, C.hd, C.train_t, C.train_e, C.test, \nC.opt_t, True)
    E.x[0] = x_max
    E.y[0] = y_max
    E.delta[0] = delta_max
    E.eta[0] = eta_max
    E.rho[0] = rho_max
    if (counter == 0):
        E.x[0] = searchlist[n]
    if (counter == 1):
        E.y[0] = searchlist[n]
    if (counter == 2):
        E.delta[0] = searchlist[n]
    if (counter == 3):
        E.eta[0] = searchlist[n]
    if (counter == 4):
        E.rho[0] = searchlist[n]
sigma_num = 0.0
sigma_den = 0.0 + C.epsilon # = small
for n in range(0, C.opt_t):
    E.rta[0] = C.ropt[n]
    E = layer3(E, False)
    if (E.rt[0] < 0.0):
        sigma_num += E.rt[0]**2
    else:
        sigma_den += E.rt[0]**2

utility = C.utilityfactor * (1.0 - C.nu) * E.pft / (float(C.opt_t - C.train_t) - C.nu * \
    sigma_num / sigma_den)

if (counter == -1 and siter == 0):
    utility_max = utility

print "%s %i %7.3f %7.3f %7.3f %7.3f %7.3f %s" % (":: Utility of parameters", E.x[0], E.y[0], \
    E.delta[0], E.eta[0], E.rho[0], ":", utility, ":")

if (utility > utility_max):
    utility_max = utility
    #store important values in container C
    C.at = numpy.copy(E.at)
    C.bt = numpy.copy(E.bt)
    C.f = numpy.copy(E.f)
    C.df = numpy.copy(E.df)
    C.dftdvt = numpy.copy(E.dftdvt)
    C.dftdwit = numpy.copy(E.dftdwit)
    C.v = numpy.copy(E.v)
    C.w = numpy.copy(E.w)
    C.x = numpy.copy(E.x)
    C.y = numpy.copy(E.y)
    C.delta = numpy.copy(E.delta)
    C.eta = numpy.copy(E.eta)
    C.rho = numpy.copy(E.rho)

print "%s %7.3f %s" % (":: Utility_max =", utility, ":")

siter = siter + 1

#Counter k + 1
C.k = C.k + 1

#no shift of data needed

if (C.i == C.train_t):
    print ":: Start of out-of-sample predictions ::"

if (C.i + 1 >= C.train_t):
    C = layer2(C)
#Counter i + 1
C.i = C.i + 1

return C

# Actual program
continue = True
if continue:
    nwights = int(sys.argv[1])
    print "%s %d %s" % (">>> Starting with ", nwights, "weights...")
    C = Container(nwights, 3, 1.0, 2000, 10, 5000, 10000, True)

print "chksum shift": numpy.sum(shift(numpy.array([0.02, 0.03, 0.04])))
print "chksum shiftf1": \\
    numpy.sum(shiftf1(numpy.array([0.02, 0.03, 0.04]), 10)), "(ref = 0.09)"
print "chksum tanhf": tanhf(1.0, C.hd, 0) + tanhf(1.0, C.hd, 1)
print "chksum tanhf": tanhf(1.0, C.hd, 0) + tanhf(1.0, C.hd, 1)

#1's
C.nu = float(sys.argv[2])

#2's (starting values)
C.x = numpy.array([50.0], float)
C.y = numpy.array([0.0], float)
C.z = numpy.array([100.0], float)

#3's (starting values)
C.delta = numpy.array([0.0], float)
C.eta = numpy.array([0.2], float)
C.rho = numpy.array([0.05], float)

data = []

file = open(sys.argv[3], 'r')
lines = file.readlines()
for j in range(1, len(lines)):
    line = lines[j].split()
    if (float(line[2]) != 0.0):
        data.append(float(line[3]))

output = open(sys.argv[4], 'w')
for j in range(len(data)-1):
    C.rta[0] = (data[j+1] - data[j])*1000.0
    C = layer3(C, True)
    if (not C):
        break
Source code of the ARL trading agent

break
print >> output, "% s % i % s % 7.3f % s % 7.3f" \\
(":X: Cumulated profit at", j + 1, "data points :", C.pft, \\
"::", C.rt[0])

#Initial testing of the ARL trading agent
#
#fx in pips!
#fx = numpy.array([0.0, -1.0, 2.0, 0.0, 1.0, 3.0, -1.0, -2.0, 0.0, 0.0, \\
# -1.0, 1.0, 0.0, -1.0, 2.0, 1.0], float)
#for j in range(0, 60000):
# fx = shift(fx)
# C.rta[0] = fx[0]
# C = layer3(C, True)
# if (not C):
# break
Bibliography


BIBLIOGRAPHY


