André Christoffer Andersen (IØT)
Stian Mikelsen (IDI)

Algorithmic Portfolio Optimisation using Evolutionary Algorithms on Neural Networks and Supporting Classifiers

Specialisation project, autumn 2011

Subject	TIØ4500 - Managerial Economics and Operations Research, Specialization Project
Student	André Christoffer Andersen
Supervisor	Professor Alexei A. Gaivoronski
Faculty	Faculty of Social Science and Technology Management
Department	Department of Industrial Economics and Technology Management (IØT)
Subject	TDT4501 - Intelligent Systems, Computer Science, Specialization Project
Student	Stian Mikelsen
Supervisor	Professor Helge Langseth
Faculty	Faculty of Information Technology, Mathematics and Electrical Engineering
Department	Department of Computer and Information Science (IDI)

Abstract

The goal of this project is to design an algorithmic portfolio optimisation system able to outperform the market portfolio. Scientifically this translates into consistently generating better return than the capitalisation-weighted buy-and-hold strategy on the securities made available to the system.

The system is designed using a three-layer modular data pipeline to transforms relevant market data into portfolio distribution advice. The first module transforms raw data into financial indicators and other basic features. These are passed onto the second module that has predictor agents (GARCH, ARIMA, RNN, SVM) which employ different time series prediction techniques to forecast market development. The third and last module contains portfolio agents (RNN, Elman, EA) which generate multiple portfolio suggestions, of which the portfolio selector (hedge, EA, follow-the-leader) selects one, or combines a subset of portfolios, for final output. Then, finally, the expected profit of rebalancing, adjusted for transaction costs, is calculated before the final portfolio is allowed to be executed.

Where applicable, the different agents are trained with evolutionary algorithms, backpropagation and stochastic gradient decent. The system is designed to be robust and adaptive. Something which is achieved by using biologically inspired methods in conjuration with financial considerations and a learning scheme design to avoid overfitting.

Preface

This report is the end product of the specialisation project of André Christoffer Andersen and Stian Mikelsen. The project lays a foundation for their Master's of Science (siv.ing) theses at the Norwegian University of Science and Technology (NTNU). The project lasted from August to December 2011.

André Christoffer Andersen is a Master's student of the Department of Industrial Economics and Technology Management (IØT) specialising in Applied Economics and Optimisation. Stian Mikelsen is a Master's student of the Departments of Computer and Information Science (IDI) specialising in Intelligent Systems.

Exploring the fringes of the interdisciplinary field of algorithmic portfolio optimisation has been both challenging and enlightening. If applying knowledge and methods from investment theory, optimisation and machine learning has taught us anything, it is that it requires a supporting and dedicated thesis committee. The authors would like to thank *their* thesis committee for the invaluable feedback and open doors they have offered in this ambitious endeavour.

The thesis committee consists of Alexei Gaivoronski, professor of finance and optimisation at IØT; Helge Langseth, professor of intelligent systems at IDI; Anders Kofod-Pedersen, associate professor of intelligent systems at IDI.

Thank you.

Trondheim, December 14, 2011

André Christoffer Andersen	Stian Mikelsen
Master's Student at IØT	Master's Student at IDI

Contents

List of Figures

List of Tables

Chapter 1

Introduction and Overview

1.1 Introduction

Algorithmic portfolio optimisation is an interdisciplinary endeavour intersecting many sciences. This field is approached from the domain of Investment Theory, Optimisation and Machine Learning.

Machine Learning is the science of autonomous decision making based on empirical data. **mathematical optimisation** is the mathematical approach for finding optimal solutions given a set of alternatives. Applying this to the problem domain of **investment theory**, the science of investment allocation, a comprehensive algorithmic portfolio optimisation system is designed.

The aim of this paper is to make an autonomous trading system based on historical data without neglecting the fundamental principles of modern portfolio theory. This paper argues that the market tends to be efficient, but given the right application of modern predictive tools it is possible to uncover underlying patterns that are too obscure for the general market to take into account.

Using biologically inspired heuristics, e.g. evolution and neural networks, in conjuration with practical classifiers and time series predication methods this paper aims to design a robust and adaptive system able scale to and outperform the financial market it operates in. The project is thus engineer-oriented rather than scientifically descriptive, though scientific principles are applied for evaluation.

1.2 Background and Motivation

1.2.1 The Problem Domain

The overarching problem this paper aims to solve is to design an autonomous trading system that can consistently generate higher return than the market it operates in.

Speculative trading in an effort to "beat the market" is as old as open markets themselves. Active investment management is now a huge business involving trillions of dollars world-wide. While being a well-established industry it is not without its share of misconceptions and controversy. The main motivation behind active investing is the prospect of generating as high of a return as possible given an acceptable level of risk. Traditionally, human traders tend to use subjective approaches for portfolio selection where there are no universally agreed upon best practice.

The advent of quantitative trading and later algorithmic trading has done little to settle the controversy. The appeal of automated trading systems that can replace emotionally constrained human financial analysts has motivated many a research project. There are numerous reasons for wanting to automate trading. One can achieve more trades, faster and with less human biases. The brunt of more advanced commercial systems focus on **high frequency trading** using short portfolio holding periods. Some of these focus on acquiring market data first in order to reap the benefit of information superiority, a practice called **low-latency trading**.

In general there are now many machine learning methods that have shown reliable results on stochastic problem domains. Especially promising are hybrids that combine multiple methods. However, as with classical probability theory[109] in ancient times[1], trading methods today are highly propitiatory. The proprietary nature of the financial problem domain makes it hard to find successful benchmark trading algorithms for evaluation. If anyone actually achieves profitable results there is no incentive to share them, as it would negate their advantage.

The whole foundation of active investing is criticised by some theorists claiming that markets are efficient and thus act like mere "random walks"[40; 41]. However, others point to underlying psychological patterns which could justify an active investment strategy [14]. For any system to actually work there must be some degree of market inefficiency. This is discussed in depth in 2.1 Financial and Investment Theory.

1.2.2 The Methodology

There are two main paradigms of asset analysis, fundamental and technical. In short, **fundamental analysis** looks at qualitative features of an asset in order to assess its intrinsic value. **Technical analysis** relies the price data only and is thus a pure quantitative measure based on historical price data. Traditionally, most trading algorithms have focused on the latter. Both methods are widely used in practice, often in conjugation with each other.

Our intention is to utilise concepts from biological systems in order to create an information ecosystem that merges the two mentioned analysis disciplines. The basic organism of this ecosystem is the predictor and investor agents. "An agent is a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its design objectives." [113, pg. 15]. By using agents the system achieve autonomy in these core computational units. The investor agent's design objectives will reflect an effort to maximise return while controlling risk. The predictor agents will focus on making time series forecasting. These are non-trivial problems that require many technical and financial questions to be resolved.

Rather than being fast, this system is intended to focus on extracting latent information in the market that is non-obvious yet valuable. To the authors' knowledge, this is an underexplored feature in applied systems. The aim is to achieve this by the use of biologically inspired methods that have shown good pattern recognition capabilities for stochastic problems with underlying latent information[12].

The investor agent is fed several streams of time dependent data, known as time series. It uses these to output a portfolio distribution for any given point in time in the series. The inner

[1]It is speculated that breakthroughs in the field of probability theory has happened several times, but never shared. This is due to its practical application in gambling.

workings of the investor agents are intended to be modular. The main focus will be on neural networks as the primary computational structure. This is due to their ability to generalise, as well as their many promising results in the financial domain[66]. Though, alternative methods will be applied as benchmarks or support.

Evolutionary algorithms will be used to set many of the meta-parameters[2] that define the agents, but some of them will be set manually when suitable. For investor agents this refers to parameters defining its data structure and learning rate. There are other important global meta-parameters like the frequency of re-balancing the portfolio. This evolutionary focus is motivated by promising result for evolving neural network architecture[115]. After evolving a population of agents the best or a combination of several well performing will be chosen. Another component in the system will decide will decide whether it is worth it to rebalance the portfolio.

The whole process will proceed in the following steps:

1. Train investor agents based on market data
2. Enhance the agent population through evolution
3. Select one or a combination of advice to form a final portfolio advice
4. Decide whether to rebalance the portfolio position

1.2.3 Personal Motivation

Intelligent systems that perform tasks autonomously is a fascinating field of computer science. The application of this field's methods to the problems deemed hardest for procedural computation is both rewarding on a personal level, and important to many fields concerned with decision support. Few problem domains are as hard for intelligent systems as making money on financial markets. A hard problem is often measured by the degree of imperfect information and stochasticity (see figure 1.1). Modern portfolio theory, with its efficient market hypothesis (EMH), states that financial markets have perfect information revealed in the market price. This reduces the problem to a find-the-efficient-frontier problem. Critics, such as financial behaviourists, say that the problem is more complex as it has elements of imperfect information and psychology[70]. This is evident in market anomalies of financial crises. Given the latter view, the problem is much more applicable to advanced heuristic and machine learning methods. The fact that this is not a settled question makes the subject interesting, and motivates the authors' effort into the problem domain.

1.3 Goals

This section will present the goals and sub-goals for this system. These form the foundation of the design process.

The ***main goal*** *for this project is to design a novel algorithmic trading system that creates optimised portfolios in order to generate a statistically significant better return than the market average.*

The main goal have been divided into three main categories of subgoals

1. System Architecture Requirements

[2]Meta-parameters: A parameter that defines an object containing (sub)parameters

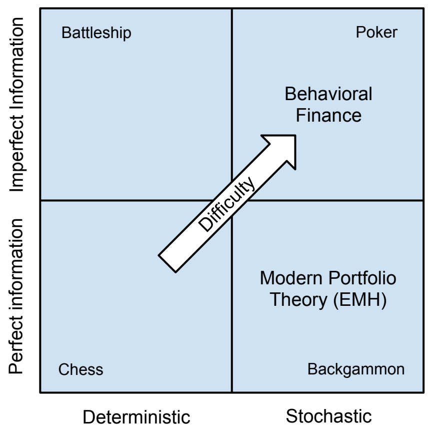


Figure 1.1: The difficulty for a computerised solution to a problem can be measured in its level of imperfect information and stochasticity. High level of both constitutes difficulty, but also opportunity.

2. Technological Feasibility and Goals

3. Financial Feasibility and Requirements

1.3.1 System Architecture Requirements

The system must handle seamless module exchange

The system should handle exchange of different combinations of technologies and methods. In order to do this seamlessly the system needs a modular architecture that allows change of approach post-compilation. This motivates the use of an architecture with an underlying general framework, rather than a single hard-coded implementation. This should increase both the testability and extensibility of the system. Modularity and testability will be the main attributes used to evaluate the architecture with. The system will consists of several interconnected components, each using different state of the art technologies. The ability to extensively test each of these components separately will be crucial to prevent complex run-time bugs.

The system should be scalable and parallelizable

Due to the inherent complexity of the system computational performance and tractability can become an issue. This can be improved be writing efficient code, as well as look at strategies for scalability and parallelization.

The system must be possible to implement in less than four months

The actual system implementation will be done as a later half-year Master's degree thesis. The architecture size and complexity must therefore take this time limit into account. This requires scheduling and setting natural milestones to evaluate progress.

1.3.2 Technological Feasibility and Goals

Identify candidate neural networks and training schemes

As mentioned the plan is to use ANNs which receive data in the form of time series and output complete portfolio advice. Which type of ANNs is best able to handle this task needs to be identified and whether the system should use several different ANNs. The possibility of including other time series prediction models will also be a part of this subgoal.

Identify method for aggregation of portfolio advice

The input found will take various forms. A central task will be to arrange input in a common and sensible format that can be analysed by the agents. In what degree this format will take time into account is also an important task. This will be closely related to the choice of ANN, more on this in the related work section 2.3.

Compose a system training scheme

The system should be robust and adaptable. The plan is to achieve this by employing learning, i.e., parameter optimization, on the system. Some of the parts like the ANN will not be feasible without learning. An important goal will be to discover what needs learning and what methods to use.

Ensure robustness and adaptivity

In order to achieve a certain credibility as a stable system it is important to have a certain degree of both robustness and adaptivity. In this paper robustness is used as the ability to handle abnormal input, while adaptivity is the ability to adapt to new environments. Robustness in this case is in regard to the market data and adaptivity to the sometimes rapidly changing financial markets.

1.3.3 Financial Feasibility and Requirements

Establish theoretical financial rationale for system validity

There are multiple questions that must be considered when it comes to the system's feasibility. One set of questions concern the financial theory and assumptions behind such a system. The market efficiency hypothesis is one of the central issues that need to be addressed.

Identify market with highest likelihood of success

The market chosen to deploy the system on is contingent on many factors. The market must be fungible[3] and liquid, yet have a degree of inefficiency that makes it possible to extract useful information to find arbitrage opportunities. Prices and market data must be readily available. Associated meta-data should be analysable and have a rich theoretical background. Finally, the complexity of the market should be low, i.e., there should be few exceptions that require manual interpretation.

Identify appropriate input data for the system

Deciding on which input data to use is of paramount importance. No matter how well a system is designed, or how good the underlying theory is, if you feed it garbage as input you get garbage as output. This is why the system needs to identify high quality information sources based on financial criteria.

[3]Fungible: Law Returnable or negotiable in kind or by substitution, as a quantity of grain for an equal amount of the same kind of grain.

Return and risk should be incorporated in the resulting portfolio advice

Though the natural aim of an algorithmic trading system is to maximise return, it is not done without perils. Securities with a high return have a tendency to entail high risk. This risk-return trade-off is an ingrained feature of modern investing. The system must therefore be able to incorporate a degree of risk aversion as well as risk reducing behaviour. The aim is to discover a hidden, more correct, risk-return relationship which the system can optimised.

Identify optimal portfolio supervisors for training the system

During training the various learning modules need some sort of benchmark portfolio to base its performance evaluation on. This portfolio is supplied by some always-correct supervisor that knows how the system ought to have acted at any given time. It is important to understand that there is no trivial way to identify such an optimal supervisor portfolio. The supervisor in itself can be taught of as a high-level parameter that should reflect the trader's risk-return preferences.

Design experiments for quantitative evaluation of the system

With systems that use black box technologies it is hard to qualitatively legitimise the advice it produces. It is therefore even more important to have a robust quantitative evaluation process that yields results that are reliable and elicit confidence.

1.4 Research Method

The research efforts are divided into three phases, a literature study phase, a system design phase and a preliminary prototyping phase.

The literature study phase delves into relevant financial theory and machine learning methods. This effort is by no means a comprehensive state of the art review, and should be seen in the context of this project's goals. Recommendations from department faculty and state of the art research reviews is used as a starting point of the literature study. The study is then expanded by a literature search based on uncovered references. Relevant articles are indexed and briefly summarised. This creates the theoretical background for the system's design decisions. The investigation into supporting technologies is done in a more colloquial manner where practical reviews and subjective experiences of expert users are considered.

In addition to the expert reviews prototyping is used to qualitatively evaluate supporting technologies and frameworks. The prototyping procedure uses a lean methodology where just enough code is produced to evaluate the technology in question. This helps discover whether or not there are hidden or practical drawback for the use of a certain frameworks or technologies.

The system design phase revolves around modelling an overarching framework that encompasses the a central data pipeline. The pipeline is a set of processing modules that transform market data into a final portfolio advice. The system architecture and data pipeline is modelled using data flow diagrams of various resolutions.

1.5 Report Structure

The remainder of this paper is divided into four chapters. Chapter 2 is Theory and Background, here the concepts and theories applied later in the report are introduced. It is the basis of the design reasoning and feasibility rational; both financial and technical material are presented. Chapter 3 is Results, here the final system design is presented and give preliminary reasoning for the individual decisions are shown and argued for. Additionally, the structure of the underlying framework, which the system design is built on, is introduced. A more thorough discussion and further arguments for the system validity can be found in chapter 4, Discussion. Finally, all is wrapped up in chapter 5, Summary and Conclusion. Here the initial goals are reviewed and mapped onto the results.

Chapter 2

Theory and Background

2.1 Financial and Investment Theory

2.1.1 Financial and Investment Theory

Financial Markets

Financial markets are mechanisms that allow people to trade liquid and fungible assets at low transaction costs. Financial markets facilitate raising capital, transfer of risk, transfer of liquidity and international trade. Capital is raised by acquiring investors. Risk can be transferred by spreading investment in uncorrelated assets. Insurance is a common example of transfer of risk between parties. There are several kinds of markets, these are shown in table 2.1. Different ones will be discussed, but the main focus is on stocks as this is the chosen market, more on this in the financial market selection section 3.1.

Stocks are instruments that represent an ownership share in the equity of a company. One can say a stock is divided into multiple shares, but for semantic purposes these words are equivalent [1]. The stock market is a public entity for the trading of such company stocks. Stocks are frequently traded on the stock market both for investment purposes and speculation. When a company earn profits they can either reinvest this in the company or choose to pay back the investors. Publicly listed companies basically have two ways to pay back profits to the stock holders. They can buy back shares or pay dividend. When a company pays dividend each stock holder receive a fixed amount per share, and the equity of the company are reduced according to the amount paid out. As stocks are connected to companies there is always the risk of them losing their value in the event of a bankruptcy. In a company bankruptcy the bank and other lenders have first claim to all remaining value, while stock holders are first paid after all debt has been settled. This makes stocks inherently more risky than to lend out money, and is also one of the main reasons one can expect more return form such investments.

Stocks listed on the market can change value drastically for many reasons concerning the company. Many of these are natural and arise from news, some are however more of a technical nature, and will create trouble for an automated trader. Two are already mentioned, dividend pay-outs and bankruptcy. Other issues may be splitting of companies, mergers or de-listing. These and dividend are no problem for ordinary stockholder as they will receive the dividend money and while their stock price drops accordingly. However an automated trader needs some

Table 2.1: This is a lists the different financial markets and briefly explains their purpose

Market	**Description**
Bond Markets	Provides financing through bonds
Stock Markets	Provides financing through stocks
Commodity markets	Facilitates trading of commodities
Money Markets	Provides short term debt financing and investment
Insurance markets	Facilitates redistribution of various risk
Futures Market	Provide standardised forward contracts for trading products
Foreign Exchange Markets	Facilitates trading of foreign exchange (currency)
Derivatives Markets	Provide instruments for the management of financial risk

way to understand that dividend is not a loss, even though the price drops.

Bonds are traditionally considered to be a low risk, low return option. These are often used as a supplement in portfolio building and as a safe haven in troubled times. Recent times have seen quite high bond yields from certain countries, but this is reflected in the sudden increase of risk these bonds are predicted to have. Still, in normal circumstances there are little return on bonds, and their main benefit would be as a safe haven.

Commodity markets differ vastly depending on which commodity is in question. Each have their own set of influential parameters that affect their price. Some of these, like gold, are used as safe havens when other more risky markets are volatile. Commodities like oil or copper are sometimes seen as a predictor of future prospects. An increase the price of copper have even been claimed to signal future rise of the stock market, though this seem to be an extinct phenomenon[3].

Forex or foreign exchange trading is trade between different currencies. This is an inherently speculative zero-sum game as all changes are relative to the other currencies. It is a popular market for speculation with many large professional actors.

Derivatives are contracts between two parties that derive their value from some underlying asset. Some common derivatives are options, futures or forwards. The reason for such derivatives are for actors to be able to decide the exact ration of risk and return and on which asset they want to “bet”. Derivatives are typically used to either speculate through certain “bets” or to hedge risk (insurance).

2.1.2 Trading

Most assets in the world are traded. Where there is trade, there can be speculation. In finance, speculation is a financial action that does not promise safety of the initial investment along with the return on the principal sum [19]. This is contrary to an investment. Speculation is often called active trading and is usually done with a short time horizon. The type of trading this system does is to be considered speculation as the plan is to generate returns by buying and selling positions in assets relatively often.

The Virtue of Speculation

A concern that might be raised about active trading is of social concern. Some claim that pure speculation is just a form of institutionalised parasitism that brings no value to the society. However, this does not take in to consideration a wider perspective. Businesses need risk capital to grow. Few investors are willing to lock their capital down for as many years as the company would need it. With a liquid financial market this can be mitigated by giving investors the ability to get rid of their exposure in the company at any given time. Speculation increases the liquidity of financial markets, thus attracting investors to growth businesses that need time to mature. In turn, this ushers forward bottom up growth and puts pressure on existing companies to stay profitable and innovative. Furthermore this increase in trading activity will in itself make the markets more efficient as more or less rational investors will correct arbitrage options they find.

Categories of Traders

There are many different types of traders active in the markets. They are often categorised by their trading time-frame. Others categorise primarily on which or what kind of strategies the traders apply. The most common will be introduced and then presented in more detail into the field of algorithmic trading and explore the field in which this project belongs.

In general there are some main distinctions to make. There is difference between technical and fundamental trading. Fundamental trading includes factors outside the price of the security, typically a company's accounts are used. Technical trading however is only based on the actual price and its qualities. Technical traders base their strategy on the assumption that historical price data can in some way predict future prices. The degrees of which analysis one employs affect the time-frame, as fundamental analysis are inherently much more static than technical. One can get a price update several times per second, while the company results are only published once a quarter. Fundamental data are also more qualitative and thus harder to analyse quickly, e.g. news. Analysis of technical data are purely quantitative, and can thus be handled by computers. Generally the shorter time-frame, the more automatic are the traders. Additionally the risk is generally higher as the market converges toward a zero-sum game when the time-frame considered decreases[1].

Position Trading The longest time-frame is the often called position trading. It generally consists of all positions held longer than a few months. This is a very reputable strategy, recommended by Warren Buffet himself[95], but with a limited possibility of return. Since one has plenty of time to do a thorough analysis of the asset, algorithmic methods have less advantage here, but can be used as supplementary guidance. This category also includes more active investors that take an active part in managing the company they invest in, like private equity companies.

Swing Trading Positions held over days or weeks include strategies like swing trading and more short-term fundamental analysis strategies. This is generally the shortest time-frame one finds purely fundamental analysis, as changes in price on a lower scale than days are generally considered noise in that perspective. Fundamental analysis is also time consuming as computers are still not fully capable of this, thus requiring a minimum time frame in this order of magnitude. There are some technical analysis done over several days, but it is limited as the primary goal of pure technical analysts is to speculate on price swings, and there are more than enough volatility

[1]The long term increase will only apply if one holds a position for some time. Trades where one aims to hold a position as short as possible will thus in principle be a zero-sum game as one trades against a snapshot in time where all values are fixed.

on a shorter time frame to do this due to the fractal[2] nature of the stock market.

Intraday Trading Day trading is trading done intraday. This is mainly speculation and mostly done by technical analysis, though one might supplement with fundamental techniques. Day trading can be a job for human investors, but require much time and are inherently risky as it is quite close to a zero sum game. Intraday trading is becoming the realm of machines as they can search through much more information. Most machines are however trading on more or less instant strategies where the aim is to hold an actual position as short as possible; these are called High-frequency traders.

High-Frequency Trading High-frequency trading (HFT) is a poorly defined term. In this paper we use it in its semantic meaning of all trading done at a high frequency instead of meddling with all the contradictory definitions in the literature. With this meaning of the term, HFT will be a broad that include all trading where the trading speed is used as a competitive advantage. In 2010 these systems trade on milli- and microseconds[54]. All HFT is done by algorithms, as the speeds required for this trading type are beyond human capabilities.

Low-Latency Trading Low-latency trading is a subclass of HFT, and can be thought of as a specific strategy of HFT. These are a naive set of simple algorithms which sole advantage is their ultra-low latency, direct market access. An example of a low-latency strategy is to survey the news. The instance good news hit the market, a low-latency trader will buy from slower sellers which have not had time to increase their price, and then sell to the new higher price. Most of these tactics revolve around finding and exploiting arbitrage possibilities fast and there is tough competition for having the fastest algorithms and the lowest latency. This activity is very capital intensive due to the hardware requirements and it needs a certain initial capital. There are few amateurs in this area and most actors are large banks, funds or even specialised algorithmic trading firms[20].

2.1.3 Efficient Market Hypothesis

The **efficient market hypothesis** (EMH) was first developed by Fama in his PhD thesis during the 1960s [39; 41]. The hypothesis asserts that financial markets are informationally efficient, in other words, prices reflect all relevant information. This means that changes in price are reactions to unpredictable events, i.e., price movements are nothing more than random walks. The EHM assumes that all actors in the market act rationally according to Expected Utility Hypothesis [107].

There are three major versions of EMH:

1. Weak EMH claims that prices on traded assets already reflect all past publicly available information.
2. Semi-strong EMH claims both that prices reflect all publicly available information and that prices instantly change to reflect new public information.
3. Strong EMH additionally claims that prices instantly reflect even hidden or "insider" information.

The efficient market hypothesis was widely accepted until the 1990s when behavioural economics started to gained ground. Empirical analysis has consistently found faults with EHM [9; 16; 88].

[2]With fractal means that whatever resolution one uses, the a graph of the price over time will always be of roughly the same volatile.

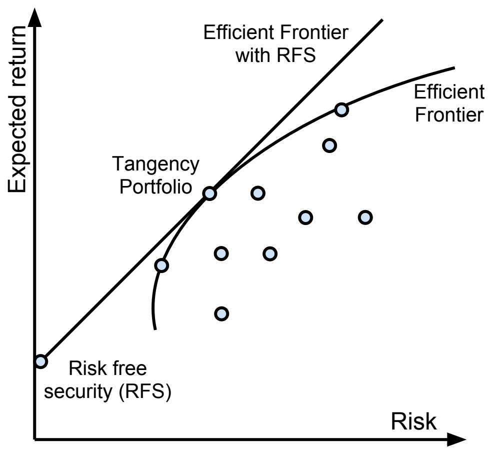


Figure 2.1: The Efficient Frontier

In spite of consistent substantial inefficiency and controversy it is still widely considered a valid starting point [17]. The EHM laid the foundation for modern portfolio theory.

2.1.4 Modern Portfolio Theory

MPT is a theory of investment that revolves around the two parameters risk and return. One can maximise the portfolio expected return given a certain risk, or equivalently minimizing risk for a given level of return. Modern portfolio theory (MPT) was introduced my Harry Markowitz in 1952 [79]. Markowitz also introduced the **efficient frontier**. The efficient frontier is the hyperbola one can draw between the set of efficient portfolios in a risk-return graph. Efficient portfolios are portfolios with the highest risk-return ratio, thus these stochastically dominate all other assets with the same risk or return. If one draws the security market line there will be a point of tangency, the portfolio in this point is called the tangency portfolio. If one then have a risk-free asset one can include in the portfolio the efficient frontier will be the straight line drawn from this risk-free rate through the tangency portfolio, see figure 2.1.

This line of optimal risk-return ratio is what Harry Markowitz coined the efficient frontier [79]. MPT formulates mathematically how one can diversify when investing. Diversification is the concept of reducing overall risk by investing in a collection of securities that are negatively correlated, thus ending up with less risk than the individual securities. MPT models an asset's return as normally distributed, defines risk as the standard deviation of this return and models a portfolio as a weighted combination of assets. MPT assumes that investors are rational and that markets are efficient.

MPT has in recent years been criticised of not matching reflecting the real world. The most central challenging field is behavioural finance. The problem of finding a portfolio on the efficient frontier can be expressed as a quadric programming problem(2.1)

$$\begin{array}{lll} \min_w \;\; \sigma_p^2 & = & \sum_{i=1}^{n}\sum_{j=1}^{n} w_i w_j \sigma_{ij} \\ \text{s.t.} \;\; \sum_{i=1}^{n} w_i r_i & = & r_p \\ \sum_{i=1}^{n} w_i & = & 1 \end{array} \tag{2.1}$$

Here, n is the number of assets and r_p is the expected rate of return. The weights of the different securities are adjusted to minimise the risk, here measured as variance. At the same time the return is fixed to its target and the allocation must sum to one.

2.1.5 Behavioural Finance

Behavioural Finance is a part of the field behavioural economics where cognitive and emotional factors are used to explain economic decisions of actors. This field is concerned with bounds of rationality of economic agents. These behavioural models typically integrate psychology and neo-classical economic theory. In 1985 De Bondt and Thaler published their result showing that people overreact to unexpected or dramatic news events. Thus discovering substantial weak form market inefficiency, this result was later reproduced and is seen by some of the start of behavioural economics [21]. This effect is further confirmed by Kahneman & Tversky well known article about the human bias in decision making under risk. They criticise the expected utility theory model and propose their own prospect theory. These prospects show several pervasive effects inconsistent with the expected utility theory. One effect prospects exhibit are to underweight outcomes that are merely probable compared to certain ones. This contributes to risk aversion in choices involving sure gains and to risk seeking in choices involving sure losses[70].

Proponents of the EMH has criticised the behavioural economics for just being a collection of anomalies. Most prominent is Fama, the father of EMH. He claims EMH is a model and as all models it does not perfectly explain the real world. He still contend that it is the best basis for investors entering the market as there are few empirical studies that have managed to capitalise on the anomalies they have found in any significant degree [38]. Others contend that experimentally observed behaviour of the kind Kahneman et al. did has little effect on the market situation as learning opportunities and competition will ensure at least close to rational behaviour [87].

2.1.6 Financial Data Measurements

This section will introduce a collection of different measurements that can be used as stock movement indicators.

Portfolio Performance Measurements

Portfolio performance is generally measured as the return achieved given the risk assumed taken. These are regarded as the two primary important aspects of any security or collection of securities. While return is easy to measure there is more discord among scholars of how to measure the risk. The traditional measure that was used by Markowitz when he presented modern portfolio theory is the standard deviation. This is a crude measure which has been criticised for treating profit and loss equally. In response to this there have been developed alternative risk measures which tries to measure only downside risk, such as Value at Risk (VaR)[68]. The VaR value represent the most one can expect to lose within a given confidence interval over a given time period. It may say something like, "We will not lose more than 4% in the next week, given a 99% confidence interval". The VaR can be used as the risk measure to calculate the efficient frontier [49].

The Beta is defined by Sharpe[96] as the covariance of the return from some collection of securities and the return from a specific security divided by the variance in return from the collection. This is expressed in (2.2) where r_a is the return of the asset a and r_p is the collection p. Normally the collection used is the whole market and the beta thus gives information of a securitie's volatility

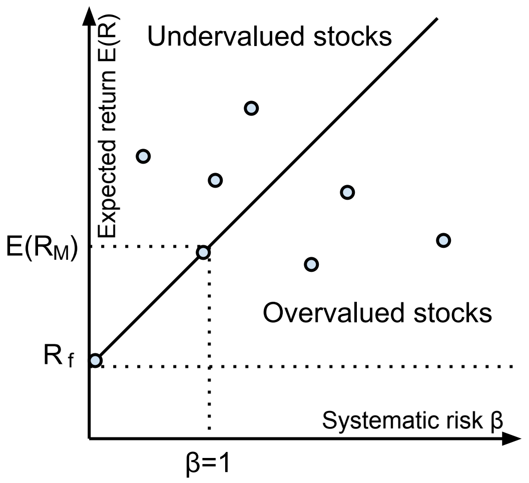


Figure 2.2: The security market line.

relative to the market.

$$\beta_a = \frac{\mathrm{Cov}(r_a, r_p)}{\mathrm{Var}(r_p)} \tag{2.2}$$

The beta is a key parameter in Sharpe's **capital asset pricing model** (CAPM). The beta originated from linear regression analysis of the returns of a portfolio versus a single security. This line is called the security characteristic line (SCL), see equation (2.3).

$$r_{at} = \alpha_a + \beta_a r_{mt} + \epsilon_{at} \tag{2.3}$$

α_a is called the assets alpha and β_a is the asset's beta. Both play an important role in MPT. The Capital Asset Pricing Model (CAPM)(2.4) is a model devised by William F. Sharpe[97] used to determine the theoretically appropriate rate of return of an asset given the assumptions of MPT. Thus this work built on Harry Markowitz's work and created an easy way to validate the MPT.

$$E(r_a) = r_f + \beta_a(E(r_m) - r_f) \tag{2.4}$$

In the equation (2.4) $E(r_i)$ is the expected return of the asset a, r_f is the risk-free rate and r_m is the market rate. It predicts the return of an asset to be the risk-free rate and the beta multiplied with the risk premium of the market. Thus it finds the risk premium for the asset.

There are several indicators created that account for both risk and return. The most common is the Sharpe, Treynor and Jensen ratios.

Treynor was the first to create such a measure, and suggested splitting risk in market risk and individual security risk. He introduced the security market line which defines the relationship between the portfolio returns and the market rates of return. The volatility between portfolio returns and market returns is represented by the beta[47].

He The Treynor ratio T is defined as:

$$T = \frac{r_p - r_f}{\beta} \tag{2.5}$$

Where r_p is the return of the portfolio p, r_f is the risk-free rate and β is the beta.

William F. Sharpe introduced the Sharpe ratio in 1966[98], this is close to identical to Treyor with the beta switched for the standard deviations. The Sharpe ratios thus incorporates all risk, not just the systematic one, see equation (2.6). This ratio is closely linked to his CAPM for which he won a Nobel prize together with Markowitz[2].

$$\frac{r_p - r_f}{\sigma_p} \tag{2.6}$$

Jensen's alpha (2.7) is also based on CAPM[64]. This measure calculated the excess return that a portfolio generates over its expected return. Thus this ratio measures how much of a portfolio's return is attributable to the manager, adjusted for market risk. This makes it a good measure of how well the portfolio as a whole has performed historically, while incorporating risk.

$$\alpha = r_p - r_f + \beta(r_m - r_f) \tag{2.7}$$

The common argument against these traditional measures is their risk measure which all incorporates both upside and downside risk. Some measures that take this aspect into account will now be presented.

Frank A. Sortino modified the Sharpe ratio to account only for downside risk in his Sortino ratio(2.8)[45].

$$S = \frac{r_a - r_f}{\left(\int_{-\infty}^{r_f}(r_f - x)^2 f(x)dx)\right)^{1/2}} \tag{2.8}$$

$f(x)$ is the probability density function of the returns. The Sortino ratio can be thought of as the excess of the investor's return per unit of downside risk, or over performance divided by the root-mean-square of underperformance.

In 2002 Keating and Shadwick published their paper criticising the use of mean and variance and proposing their own performance measure, the omega[71]. The problem with all measures based on mean and variance is that they cannot capture all interesting risk and return features unless the returns are normally distribution[72]. For an example of their critique on mean and variance see figure 2.3.

$$\Omega(r) = \frac{\int_r^{\infty}(1 - F(x))dx}{\int_r^{\infty} F(x)dx} \tag{2.9}$$

Translated to English it reads the ratio of the probability of having gain by the probability of having a loss.

Security Performance Measurements

So far we have looked at measures of complete portfolio performance. In order to build a good portfolio the system will need individual security measurements measurement as well as indicators. These will together be the foundation for the analysis which the portfolio selection is based on. This section will list the indicators chosen. This is based on reading other papers in addition to some theories of the authors' own. Many of these are old rules of thumbs or indicators that have previously been shown to identify anomalies. While the return is easy to measure, it can be very hard to predict. Many of these measurements are meant to be able to indicate future returns, often by evaluating whether a company is under or overvalued. The indicators are divided into fundamental and technical ones. The technical are derived only from the price

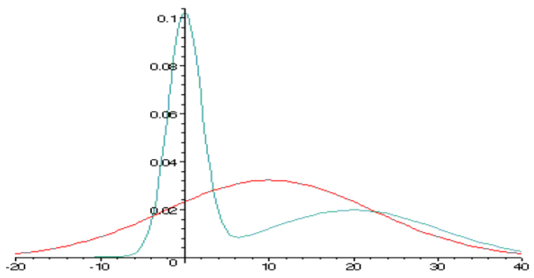


Figure 2.3: These distributions have the same mean and variance. Keating and Shadwick developed the omega to be able to more accurately capture the important features in time series, rather than depending on variance and mean.

Table 2.2: Fundamental Financial Indicators

Feature	Description	Formula	Reference
P/E	Compares stock price to company earning. Gives indication of over-/under valuation	$\frac{\text{Price}}{\text{Earnings}}$	[28; 34]
P/B	Compares market stock price with the accounted price, book value. Another indication of over-/under valuation	$\frac{\text{Price}}{\text{Book Value}}$	[28]
PEG	Compares P/E to the companies' growth. Estimates value while accounting for growth	$\frac{\text{P/E}}{\text{Annual EPS Growth}}$	[34]
Beta	The relative variance of a stock a compared to the market's m variance.	$\frac{\text{Cov}(r_a,r_p)}{\text{Var}(r_m)}$	[96]
Volume	The number of shares traded in a security during a given period of time		

itself, while fundamental use external factors. Many of the fundamental are partly made up from accounting data. These are usually updated at most once each quarter and will thus do a leap each time new accounts are published. An example of this is the Price-Earnings-ratio (P/E).

Market Performance Measurements

Another set of indicators which may be of interest are macro indicators and period indicators. These are factors that are not directly related to any single security, but still considered to affect the stock market indirectly. Many of the indicators may seem trivial, but it is important to include factors like data, such that the system can use it to identify patterns. In table 2.4 the general categories of indicators which can be used have been listed. Together these are meant to provide an important context to the fundamental and technical indicators as well as providing a general measure of the state of the market altogether.

Table 2.3: Technical Financial Indicators

Feature	Description	Formula	Reference
ROC	Rate of price change. Difference in price from n time steps ago	$\frac{C_t}{C_{t-n}}$	[86]
RSI	Relative strength index. Ranges from $[1, -1]$	$1 - \frac{1}{1+\frac{\sum_{i=0}^{n-1} \frac{Up_{t-i}}{n}}{\sum_{i=0}^{n-1} \frac{Dw_{t-i}}{n}}}$	[4]
Williams' %R	Momentum indicator that measures overbought/oversold levels	$\frac{H_n - C_t}{H_n - L_n}$	[4]
Disparity n	n-day disparity. The distance between the current price and n-day moving average	$\frac{C_t}{MA_n}$	[29]
%K	Stochastic %K. Compares where a security's price closed relativt to its price range over a given time period. Here LL_t, HH_t is lowest low and highest high, respectively.	$\frac{C_t - LL_{t-n}}{HH_{t-n} - LL_{t-n}}$	[4]
Momentum	Price change over a given time span	$C_t - C_{t-n}$	[62]

Table 2.4: General and Macroeconomic Indicators

Feature Extractor	Description
Periodic Time Indicators	Cyclical time counters that tell the system what period it is in. Possible indicators are the month of year, day of month, day of week and trading hour.
Economic Indicators	A collection of low resolution indicators published by reputable sources. Possible indicators are unemployment rate, gross national product and consumer price index.
Market Indecies	Aggregated price development of the market in question, i.e., composite index. Additions to this can be sector specific sub-indices, e.g., technology or health care indices in a stock market.

2.2 Technology Methods

2.2.1 Data Mining Concepts

The following definitions within machine learning and data mining are based on Elkan's 2011 lecture notes on "Predictive analytics and data mining"[36].

Data mining is an interdisciplinary field that applies statistical methods and learning algorithms to real-world datasets. It is a goal oriented discipline with a focus more on creating applicable predictions to real-world problems rather than uncovering how the underlying system works. This makes data mining more of a predictive science rather than a descriptive one. Machine learining and data mining terminology will be used for the technical description of the system. The following is a brief introduction to the concepts and terminology of data mining and machine learning.

Data Structure

Machine learning typically define a piece of data as a **data point**, instance, record or example. This paper will refer only to data points. Most often data points are fixed size vectors of real values. A data point indexed by i will give $\boldsymbol{x}_i = [x_{i1} \ \cdots \ x_{in}]^{\mathrm{T}} \in \mathbb{R}^n$. When you have a set of data points they are referred to as a **data set**. By convention a data set is often illustrated and stored as a matrix where each data point is a row, $X = [\boldsymbol{x}_1 \ \cdots \ \boldsymbol{x}_m]^{\mathrm{T}} \in \mathbb{R}^{n\times m}$. The individual values of the data set X have the structure of (2.10).

$$X = \begin{pmatrix} \boldsymbol{x}_1^{\mathrm{T}} \\ \vdots \\ \boldsymbol{x}_m^{\mathrm{T}} \end{pmatrix} = \begin{pmatrix} x_{11} & \cdots & x_{1n} \\ \vdots & \ddots & \vdots \\ x_{m1} & \cdots & x_{mn} \end{pmatrix} \tag{2.10}$$

A column in a data set is referred to as a **feature**, the equivalent concept in a data point is called a feature value. This means that there are n features in a data set. When referring to a feature in a data set this refer to a common attribute that describes something about each of the data points. If a data set describes a collection of cars, then two of the feature could be "top speed" and "mileage". The collection of features that describe the attributes of data set is called a **feature set**. In practice, a data point can be anything from a customer record to a sensor reading. A collection of time dependent readings is often referred to as a **time series**. This makes features and time series directly analogous. A vector containing readings from several time series at one point in time would thus constitute a data point. Time series predictions about the future is called **forecasting**.

Prediction using Supervised Learning

Prediction is in essence a process of trying to estimate some unknown feature associated with a data point. This feature is called a **label**. There can be many labels for one data point $\boldsymbol{x}_i$ such that $\boldsymbol{y}_i = [y_{i1} \ \cdots \ y_{ik}]^{\mathrm{T}} \in \mathbb{R}^k$. This gives a label structure for Y equivalent to the data set X. This is illustrated more succinctly in 2.11.

$$Y = \begin{pmatrix} \boldsymbol{y}_1^{\mathrm{T}} \\ \vdots \\ \boldsymbol{y}_m^{\mathrm{T}} \end{pmatrix} = \begin{pmatrix} y_{11} & \cdots & y_{1k} \\ \vdots & \ddots & \vdots \\ y_{m1} & \cdots & y_{mk} \end{pmatrix} \tag{2.11}$$

In practise the label tends to be the optimal action given the current state, but can also just be a prediction that needs some further analysis for decision making. Inferring the "true" labels of unlabelled data points based on a discretely labelled data set is done by a **classifier**. However, when intentionally being ambiguous, this paper refers toboth prediction models and classifiers under a collective term, **predictors**. For single data points, a predictor f can be defined as in (2.12), where $\boldsymbol{x}_i$ is defined as above, $\hat{\boldsymbol{y}}_i$ is a prediction of $\boldsymbol{y}_i$ and $\boldsymbol{\theta}$ is a set of parameters that defines the classifier. This notation extends to full data sets, see (2.13), where $\hat{Y} \in \mathbb{R}^{n\times k}$ is defined as in (2.11).

$$f(\boldsymbol{x}_i; \boldsymbol{\theta}) = \hat{\boldsymbol{y}}_i \tag{2.12}$$

$$f(X; \boldsymbol{\theta}) = \hat{Y} \tag{2.13}$$

A predictor takes a data point as input and outputs one or more labels. Prediction using multiple labels is called multi-labelled prediction or classification. This is accounted for in the label definition above. As indicated in (2.12) and (2.13), a predictor can be seen as a function that tries to map data points to labels. In those terms you can view the values which the input data points can take as the domain of a classifier. Conversely, the values that the labels may take can be viewed as the range of a classifier. In the case of discrete labels one use the term class rather than value, and talk about **classification** rather than prediction.

Predictors can employ learning algorithms based on numerous machine learning and statistical methods. When a predictor learns its parameters $\boldsymbol{\theta}$, or weights, one call this **training** the predictor on a data set. In this project the emphasis is on **supervised learning** and a variation called **reinforcement learning**.

Supervised learning requires feedback about the performance of the predictor for it to be able to learn. Typical supervised learning indicates the error of a prediction. This can be used directly for learning. The training process indicates, in relative terms, how much and in what direction to improve the predictor's parameters. Examples of supervised learning is stochastic gradient following on a linear regression problem.

Reinforcement learning, on the other hand, gives only utility as feedback. That is, only information about how good a solution is, not how to improve it. The utility can be any measure that indicates the performance of an action, or series of actions, in a given problem state. Evolutionary algorithms typically operate under these constraints.

Performance Evaluation and Overfitting

One critical issue with machine learning is the danger of **overfitting**. Overfitting happens when a predictor learns the inherent noise of the data set. Making a classifier overly complex, e.g., giving it too many parameters, can make it able to "fit" its parameters exceptionally well to the data set it is learning from. Though the classifier can give up to perfect accuracy on this data set, it is not able to generalise to new data points. This has repercussions with how one should evaluate a classifier's capabilities. A common way to uncover overfitting is to split the labelled data set into two separate data sets. One is called a **training set** the other a **test set**. The larger training set is used to train the classifier, while the smaller testing set is used to test it. It is important to hold the two data sets entirely separate and to use the test set only once in order to prevent **information leakage**. This can occur when a classifier gets information from the training set inadvertently. Having a test set is good practice for knowing when one is overfitting, however, how to avoid it is not as simple. Typically you would try to limit the complexity of the classifier such that it is not capable of learning noise. For some predictors it is possible to measure when overfitting starts while learning. This is done with a third **validation set** which the predictor tests on while training. When the results on the validation set starts to deteriorate training is terminated.

2.2.2 Regression methods

This section will present different regression models planned to be used in the system. Details of their part in the system will follow in the results part. To motivate the following methods we will first look into linear regression, then move on to the autoregressive methods ARCH and GARCH. Note that the following models will be presented in a data mining context, thus omitting noise variables. This is because all the models are in terms of the actually estimated values and not true value equals to noise added to an estimate. Though, the noise in data mining would be equivalent to error of prediction.

Linear Regression

Linear regression tries to learn a vector of weights $\boldsymbol{\theta}$ that are linearly combined with data point $\boldsymbol{x}_i \in X$. As previously described, a classifier can output multiple, specifically k, labels $y_{ij} \in \boldsymbol{y}_i \in Y$ for some data point $\boldsymbol{x}_i$. However, the classical linear regression classifier outputs only one single real valued label, such that $\boldsymbol{y}_i = y_i \in Y = \boldsymbol{y} = [y_1 \ \cdots \ y_m]^{\mathrm{T}}$. A linear regression classifier can thus be defined as (2.14). An intercept, or constant term, is often used in linear regression. Without an intercept a linear regression model is forced to intersect the origin. Adding an intercept while making the mathematics compact can be done by augmenting the training set with a feature of only ones, i.e., the one vector $\mathbf{1} \in \mathbb{R}^m$, such that $X' = [\mathbf{1} \ X] \in \mathbb{R}^{n \times m+1}$. This operation is called a **one-augmentation** of the data set X. Performing this operation requires that the number of weights in $\boldsymbol{\theta}$ be increased by one, yielding $\boldsymbol{\theta}' \in \mathbb{R}^{m+1}$. The classifier remains identical to that of (2.14), i.e., it can be used directly, $f(X'; \boldsymbol{\theta}')$.

$$f(X; \boldsymbol{\theta}) = X\boldsymbol{\theta} = \boldsymbol{y} \tag{2.14}$$

Actual learning of $\boldsymbol{\theta}$ can be done with (stochastic) gradient following using a differentiable error such as squared error. That is, an error function can be used that utilises the differentiability of the euclidean distance between prediction and labelling as in (2.15).

$$\varepsilon = ||y - f(X; \boldsymbol{\theta})||_2^2 \tag{2.15}$$

Non-linear Regression, using the Kernal Trick

The problem with linear models, like linear regression, is that there might be non-linear relations that cannot be represented linearly. Trying to fit a straight line to data set of mortality rates versus blood pressure might yield horrible results. This is because neither very high nor very low pressure is good. It would be much more reasonable to fit a polynomial function to this problem. This leads to looking for ways to transform a non-linear problem to a linear one. That is, one can translate a linear regression model in to a **non-linear regression** model by transformation of the feature space it is applied to. The following procedure is a general concept in machine learning called the **kernel trick**. In essence you push the data set into a higher dimensional non-linear space in the hope of capturing linear relations.

An example linear model could be described by $\boldsymbol{x} = [1 \ x_1 \ x_2]^{\mathrm{T}}$ and $\boldsymbol{\theta} = [\theta_0 \ \theta_1 \ \theta_2]^{\mathrm{T}}$ with classifier $f(\boldsymbol{x}; \boldsymbol{\theta}) = \boldsymbol{x}^{\mathrm{T}}\boldsymbol{\theta}$. If one transform $\boldsymbol{x}$ to some higher dimensional polynomial variant $\boldsymbol{x}' = [1 \ \ x_1 \ \ x_2 \ \ x_1^2 \ \ x_2^2 \ \ x_1x_2]^{\mathrm{T}}$ and add more parameters such that $\boldsymbol{\theta}' = [\theta_0 \ \cdots \ \theta_5]^{\mathrm{T}}$, then the classifier $f(\boldsymbol{x}'; \boldsymbol{\theta}')$ is still linear, but we're doing a non-linear classification task. The specific transformation does not need to be polynomial as in this example; any transformation can be used.

Autoregression

The purpose of using regression in forecasting is to predict the next value of a time series. **Autoregressive models** work much in the same fashion as linear regression. The difference lies mainly in how you define the input data. Normally the internal structure and correlation of a data point's feature set is not of importance, this is in contrasts to autoregressive models who have feature sets defined by a window that slides along a time series. A data point, i.e., a row of the final data set, is simply a snapshot of all the values within a window of the time series. For each snapshot the produced data point is stored and the upper and lower bound of the window incremented, finally the process is repeated. If one let a time series be defined by a vector $\boldsymbol{z} = [z_1 \ \cdots \ z_t \ \cdots \ z_T]^{\mathrm{T}}$ and use a window with size p, then (2.16) describes the transformation process needed to convert $\boldsymbol{z}$ into a data set $x_{ij} \in X$ applicable to the previously described linear regression classifier. Thus, with one-augmentation of the data set, the autoregressive model reduces to a linear regression task with a classifier defined by (2.17).

Table 2.5: A more visual illustration of the sliding window time series conversion to a data set.

$z_t = x_{ij} \in X$	$j = 1$	$\cdots$	$j = p$
$i = 1$	z_1	$\cdots$	z_p
$i = 2$	z_2	$\cdots$	z_{p+1}
$\vdots$	$\vdots$	$\ddots$	$\vdots$
$i = T - p + 1$	z_{T-p+1}	$\cdots$	z_T

$$\begin{aligned} x_{ij} &= z_t \\ t &= i + j - 1 \\ i &\in \{1, \cdots, T - p + 1\} \\ j &\in \{1, \cdots, p\} \end{aligned} \tag{2.16}$$

$$\begin{aligned} \hat{z}_{t+1} &= f(\boldsymbol{x}_i; \boldsymbol{\theta}) = \boldsymbol{x}_i^{\mathrm{T}} \boldsymbol{\theta} \\ i &= t - p + 1 \end{aligned} \tag{2.17}$$

It is possible to forecast further in to the future by creating new data points as prediction is performed. That is, when a prediction is made for one time step into the future, $T+1$, the value of this prediction can be used to create another data point $\hat{\boldsymbol{z}}_{T+1} = \hat{\boldsymbol{x}}_{T-p+2}$ with the structure described in (2.16). Now applying this result to the time series predictor can yield another time series value, now at time $T+2$. This can be carried continued as far as needed. Of course, the further you go into the future the less reliable the predictions are.

Training of the autoregression weights can be done in numerous ways. One common way is using Yule-Walker equations to find lag coefficients and solving directly by inversion. Another way is to solve the linear regression problem by maximum likelihood using stochastic[3] gradient training [35].

Moving Averages

A simple filtering mechanism for time series, sometimes used for rudimentary forecasting, is the **moving averages** method. The setup for moving averages works much in the same way as the autoregressive model just developed in (2.17). The difference being that there is no learning. The weights are fixed a priori, or determined by the development of the time series. The simplest form is the rolling mean. It simply finds the mean value of a sliding window and outputs this filter value as a prediction. This can be modelled as $\theta_j = 1/q$ where q is the window size.

ARMA and ARIMA

Autoregressive Moving Average (ARMA) models is a traditional statistical approach to predicting time series. Box and Jenkins are often credited with the original ARMA approach[23]. Today, studies with ARMA combined with different neural networks have shown promising results[116].

In simple terms, ARMA is a combination of an autoregressive model and a moving average model[25]. The autoregressive model handles learning of weights that reflect the time series values while the moving average model tries to reflect noise in the data. In (2.18) a classifier is defined g using ARMA(p,q), where p and q are window sizes. Here the time series is transform

[3]The term "stochastic" in stochastic gradient descent refers to how the learning method is performed. It is a stochastic approximation to gradient descent, where updates are performed based one data point at a time.

$\boldsymbol{z}$ in to the two data sets $\boldsymbol{u_r} \in U$ and $\in \boldsymbol{v}_s \in V$ based on (2.19) and (2.20), respectively. In order to include a constant, i.e., intercept, one-augmentation of U is required.

$$\hat{z}_{t+1} = g(\boldsymbol{u}_r, \boldsymbol{v}_s; \boldsymbol{\theta}, \boldsymbol{\phi}) = f(\boldsymbol{u}_r; \boldsymbol{\theta}) + f(\boldsymbol{v}_s; \boldsymbol{\phi}) = \boldsymbol{u}_r^{\mathrm{T}}\boldsymbol{\theta} + \boldsymbol{v}_s^{\mathrm{T}}\boldsymbol{\phi} \tag{2.18}$$

$$\begin{aligned} u_{rj} &= z_t \in \boldsymbol{u_r} \\ t &= i + r - 1 \\ r &\in \{1, \cdots, T - p + 1\} \\ j &\in \{1, \cdots, p\} \end{aligned} \tag{2.19}$$

$$\begin{aligned} v_{sj} &= \Delta z_t = z_t - z_{t-1} \in \boldsymbol{v_s} \\ t &= i + s - 1 \\ s &\in \{1, \cdots, T - q + 1\} \\ j &\in \{1, \cdots, q\} \end{aligned} \tag{2.20}$$

In (2.18) there are two sets of parameters. The trainable autoregressive parameters $\boldsymbol{\theta}$ and non-trainable moving average parameters $\boldsymbol{\phi}$.

A generalization of the ARMA model is the **Autoregressive Integrated Moving Average** (ARIMA) model. This model is simply a form of non-linear regression that can be modelled by transforming the feature set to higher dimensions. The same linear regression classifier can be applied for the autoregressive part.

GARCH

The **Generalised Autoregressive Conditional Heteroscedastic** (GARCH) model was independently developed by Bollerslev[105] and Taylor[61] in 1986.

A GARCH model is a non-linear application of ARMA using **conditional variance** as the time series, $z_t = \sigma_t^2$ [25]. An explicit formulation of GARCH(p,q) model is represented in (2.21).

$$\hat{\sigma}_t^2 = \alpha_0 + \sum_{i=1}^{q} \alpha_i u_{t-i}^2 + \sum_{j=1}^{p} \beta_j \sigma_{t-j}^2 \tag{2.21}$$

The conditional variance is a one-period-ahead estimate for the variance calculated based on past information. The advantages of GARCH compared to ARCH are that it is more parsimonious and avoids overfitting. GARCH manages to model the fact that volatility occurs in bursts which is a phenomenon observed in financial time series. GARCH is used to forecast volatility in time series [25]. GARCH will be used to give us a volatility predication that might perform better on financial series than the traditional variance measure.

SVM

Support Vector Machines (SVM) are a set of supervised learning methods used for classification and regression analysis. SVMs are linear non-probabilistic binary classifiers, i.e., they can classify data points as one of two classes. This is done by intersecting a hyperplane through the feature space that separates one cluster of similarly labelled training data from another. The SVM learns the parameters for this hyperplane by maximising the margin from the hyperplane to the two training data clusters.

More advanced SVMs will use soft margins[31] that react gracefully to abnormally labelled data points and semi-overlapping data clusters that cannot be separated by a simple hyperplane. Though, the more overlap between the two clusters of data points, the worse a SVM will do. This can be mitigated by transforming the initial feature space into a higher dimensional feature

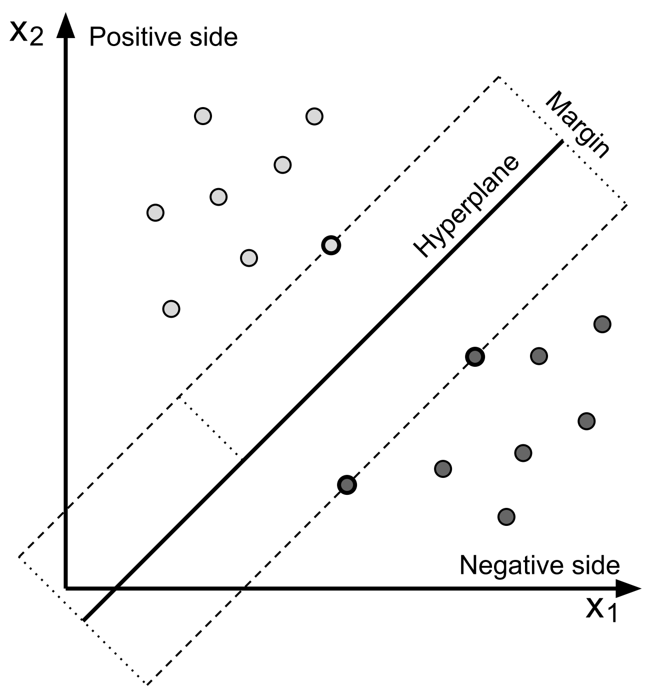


Figure 2.4: Illustration of a margin maximising SVM hyperplane, in a two-dimensional feature space, separating positive- and negatively labelled data points.

space by using the kernel trick, as described with the non-linear regression example. However, this increases the danger of overfitting, as with most models that increase the number of parameters.

The basic SVM classifier assumes data point separability. It has a one-augmented data set $X \in \mathbb{R}^{m \times n}$ of n data points $\boldsymbol{x}_i$ and m features (including the augmented feature). This follows the data set definition of (2.10), where X is a matrix with rows of data points $\boldsymbol{x}_i$. Since this is not a multi-label problem there is only one label per data point. One can therefore compound all the labels related to data set X in to one single binary vector $[y_1 \ \cdots \ y_m]^{\mathrm{T}} = \boldsymbol{y} \in \{-1, 1\}^m$ which we want to use to find the model parameters $\boldsymbol{\theta}$. The separating hyperplane is defined by $\boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{z} = 0$ for any $\boldsymbol{z} = [1 \quad z_2 \ \cdots \ z_n]^{\mathrm{T}} \in \mathbb{R}^n$. Surrounding this we want to make two parallel separating hyperplanes, or margins, which we maximise the distance between. For the two different classification we can defined the margins by (2.22), which can be combined to that of (2.23).

$$\begin{array}{rcl} \boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{x}_i & \leq & +1 \quad \text{for} \quad y_i = +1 \\ \boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{x}_i & \geq & -1 \quad \text{for} \quad y_i = -1 \end{array} \tag{2.22}$$

$$y_i \boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{x}_i \geq 1 \ \forall i \tag{2.23}$$

Because the parameter vector $\boldsymbol{\theta}$ is the normal to the separating hyperplane we have that $\frac{1}{||\boldsymbol{\theta}||_2}$ is the "width" of the margin. This means we can maximise this margin by minimizing the denominator, i.e., euclidean norm $||\boldsymbol{\theta}||_2$, subject to the separation of all the data points. This leads to the raw minimization problem of (2.24). Here, the function diag$(\cdot)$ converts a n dimensional vector into a n-by-n matrix with the vector elements in the diagonal, otherwise zero. To heuristically solve this we can cast it in to a Lagrangian relaxed quadratic optimization problem[27]. It can be shown that we are trying to find the saddle point of (2.25). The reason for finding the saddle point rather than the minimum is that the Lagrangian multipliers will otherwise get the trivial solution of $\lambda_i = \inf$. Data points $\boldsymbol{x}_i$ that have Lagrangian multipliers counterparts which are $\lambda_i > 0$ are said to be the support vectors. They are the once that define the separating hyperplane.

$$\begin{array}{rl} \min_{\boldsymbol{\theta}} & ||\boldsymbol{\theta}||_2 \\ \text{s.t.} & \text{diag}(\boldsymbol{y})(X\boldsymbol{\theta} - \mathbf{1}) \geq \mathbf{0} \end{array} \tag{2.24}$$

$$\begin{array}{rl} \min_{\boldsymbol{\theta}} \max_{\boldsymbol{\lambda}} & \frac{1}{2}\boldsymbol{\theta}^2 - \boldsymbol{\lambda}^{\mathrm{T}} \left[\text{diag}(\boldsymbol{y})(X\boldsymbol{\theta} - \mathbf{1})\right] \\ \text{s.t.} & \boldsymbol{\lambda} \geq 0 \end{array} \tag{2.25}$$

An expansion of the formulation of (2.24) can be made where a soft margins is included. It introduces a variable $\boldsymbol{\xi} \in \mathbb{R}^n$ yielding (2.26). As done with the basic version this can be Lagrange relaxed and made in to a quadratic optimization problem. Constant C is a regularization meta-parameter that must be tuned. It reflects the trade-off between maximizing the margin and minimizing the training error.

$$\begin{array}{rl} \min_{\boldsymbol{\theta},\boldsymbol{\xi}} & ||\boldsymbol{\theta}||_2 + C\mathbf{1}^{\mathrm{T}}\boldsymbol{\xi} \\ \text{s.t.} & \boldsymbol{\xi} \geq \text{diag}(\boldsymbol{y})(X\boldsymbol{\theta} - \mathbf{1}),\ \boldsymbol{\xi} \geq \mathbf{0} \end{array} \tag{2.26}$$

2.2.3 Evolutionary Algorithms

An **evolutionary algorithm** [43] is a metaheuristic optimization algorithm that draws inspiration from biological evolution. Biological evolution works through natural selection, where individuals in a population of a species compete for the possibility to reproduce. Individuals who are able to create many viable offspring are said to be fit. This means that their inheritable traits, i.e., phenotypes, will tend to be more pervasive than their less fortunate counterparts in the following generation. Over many generations this will tend to allow a population to adapt to changes in the environment, and increase its viability. Notice that changes in phenotypes does not occur in individuals, but rather between generations. This biological explanation of evolution is analogous to evolutionary algorithms used for parameter optimization.

In evolutionary algorithms an individual can be viewed as a solution method to a problem. The problem must be repeatable and give feedback about the utility of the solution method. In evolutionary terms the utility that an individual generates is called fitness. The fitness that an individual can produce is determined by its parameters, i.e., phenotype.

There are many specific ways to perform evolution, the following is a typical example. A population of randomly generated individuals is created. The individuals, i.e. solution methods, are applied to the problem at hand and fitness levels are recorded. First, a subset of the worst performing individuals is removed from the population. Then, a subset of individuals with phenotypes that yield the highest fitness is allowed to reproduce. The reproducing individuals are naturally called the parents of that generation. Reproduction is performed by creating multiple clones of the parents. The parameters of the clones are given random, yet modest, adjustments.

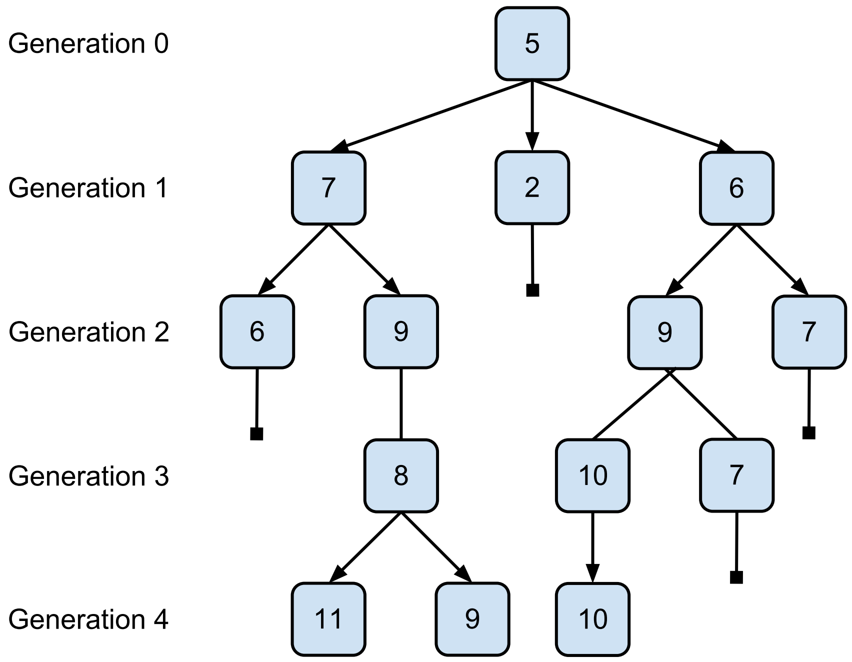


Figure 2.5: An illustration of four generations of an evolutionary algorithm was the two most fit individuals are allowed to reproduce while the rest are terminated. The phenotypes are randomly mutated with a uniform distribution over $[-3, 3]$.

This action makes them children of their parents, rather than clones. The children are added to a fresh population and another round, i.e., generation, is performed. This continues until some stopping criteria are reached. An illustration of this algorithm in practice is shown in figure 2.5.

There are many important variations of this basic evolutionary algorithm. One of the more important variations is the use of a genotype as the medium of mutation. A genotype is a discretely encoded version of a phenotype. A typical type of code could be an array of binary values. In biology, DNA is the genetic code describing a genotype. The benefit of using a genotype is that it tends to increase a populations ability to find better solutions. This is because small changes in a genotype might give a drastically different phenotype, thus nudging the solution search out of local minima.

Other important variations include using more severe mutations, like cross-over, and changing the termination or breeding protocol.

Evolutionary algorithms are good at performing many parameter adjustment tasks, like finding suitable parameters of a classifier. Though, it is even more suitable for meta-parameter optimization, i.e., optimization of the structure of a classifier rather than its lower level parameters. It is common to let evolutionary algorithms take responsibility of adjusting the proverbial "shape" of a classifier and use some other learning method to find the lower level parameters.

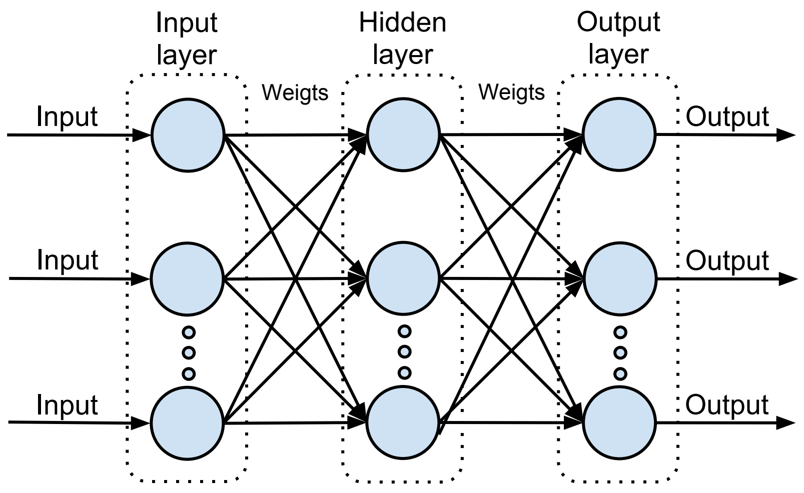


Figure 2.6: A feed forward neural network with a single hidden layer.

2.2.4 Artificial neural networks

Artificial neural networks (ANN) are computational structures inspired by the neural networks of biological brains. Artificial neural networks consist of a set of interconnected neurons that interact with each other by sending variable strength activation signals. Each connection has an associated weight, i.e. parameter, which modulates the activation signal. The activation signals from all connections to a neuron are combined by a transfer function. This function is typically just the summation of the individual weighted activation signals. The transfer function's output is passed to an activation function which fires if the output surpasses a threshold. Some implementations use an S-shaped activation function, such as a sigmoid function, instead of binary activation. The resulting activation function output is then passed along the neurons outgoing connections.

Training the Neural Network

When training a neural network you are adjusting its weights (and thresholds). This can be done with regard to some metric of performance. Typically this is done by minimization of the error between a target output, i.e., label, and the predicted output. In practice this is usually done with a specialised gradient decent algorithm such as **backpropogation**. Since gradient decent is a greedy algorithm, backpropogation suffers from the same common problems associated with this technique. It may get stuck in a local optima of multimodal solution spaces and it need a differentiable error function[115].

Aside from backpropagation there are conjugate decent algorithms, also based on gradient decent, with much the same advantages and drawbacks. To overcome the disadvantages with the gradient decent approach evolution can instead be used on the connection weights. Evolution can effectively find near optimal set of weights without computing gradient information. A disadvantage with evolution is that it cannot guarantee a optimal solution and often have problems when it comes down to fine-tuned search. Thus a hybrid solution between evolution and a gradient decent based approach is common, and shows promising results[115]. Evolution does the rough search and then may use gradient decent for the last part of the search.

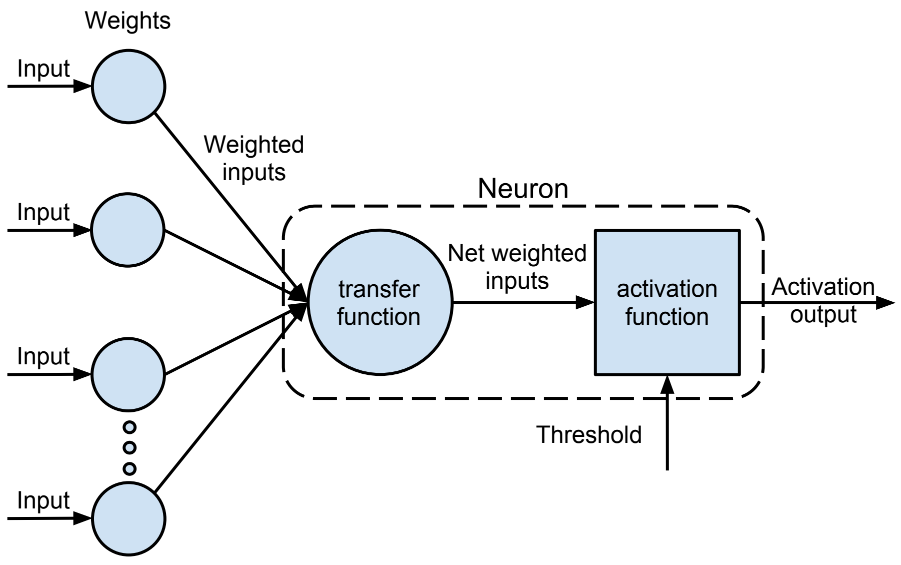


Figure 2.7: A neuron sums multiple weighted inputs from other neurons and applies it to an activation function. If the net weighted input exceeds the activation threshold, then the activation function will fire a signal known as the activation output.

ANNs are very prone to overfitting as they will continue to adjust until they perfectly fit any input data [117]. To avoid this problem it is important to stop the training when at the point where the generalisation ability starts to fall. It is also important to have a large set of data relative to the size of the network.

Feedforward vs Recurrent Network

There are two main categories of artificial neural networks, **feedforward neural networks** and **recurrent neural networks**(RNN). The difference between them is that RNNs allow for cycles, while feed-forward ANNs require the flow of signals to always go towards the output neurons. Recurrent networks have memory and attractor dynamics [112]. One of the drawbacks with recurrent networks is that they tend to have complex and often obscure behaviour and are considered harder to train and understand. Jordan[67] was the first to create such networks, but Elman's [37] adaptation is more commonly used. The main difference between feed-forward networks and RNNs is that the latter naturally takes in to account previous signals and can thus achieve memory and a "sense of time". This makes RNNs well suited to discover temporal patterns [11; 51].

Evolving ANN Architecture

When referring to an ANN's architecture this include the topology and the transfer functions of a network. This includes of how many layers are used, how many nodes in each layer has and how the neurons are interconnected. The architecture of a network is crucial for its information processing capabilities [115]. Traditionally the architecture design has been a human expert's job.

When presented with a large solution space and no known algorithm to easily create an optimal architecture it is common to use trial and error. There has been done some research toward automatic design by constructive or destructive algorithms [100]. Angeline et al. [6] points out the weakness of such approaches as they tend to end up at local optima, they also only investigate a restricted topological subset rather than the complete architectural search space [115]. More recent research on automatic architecture focus on genetic algorithms. These are less computationally intense, cover a large search space and are less prone to over-design than those designed by human experts[22]. The search space to find a suitable ANN architecture is infinitely large, non-differentiable, complex, deceptive and multimodal[115]. All these characteristics make it hard or impossible to use traditional algorithms like gradient decent. This leaves evolutionary algorithms using genetics a better chance to find good solutions to the architecture problem.

Miller et al. shows that evolution is well suited for finding neural network architectures. This is because evolution is adept at this type of search spaces[82].

- The search surface is infinitely large as one can always add more neurons
- The search surface is nondifferentiable as there are a discrete set of neurons
- The search surface is noisy since the mapping between architecture and performance is indirect.
- The search is deceptive as small changes in architecture can make large differences in performance
- The search surface is multimodal since different architectures may have similar performance

2.2.5 Hedge Algorithm

The **hedge algorithm** is a online meta-classifier developed by Freund and Schapire[48] for allocating limited resources based on advice from m classifiers. For each time t the meta-classifier allocates weights, or resources $\boldsymbol{\theta}^t = [\theta_1^t \ \cdots \ \theta_m^t] \in [0,1]^m$, to each advising classifier $j \in I = \{1, \cdots, m\}$, such that $\sum_{j \in I} \theta_j^t = 1$. The advising classifiers incur individual loss $\boldsymbol{\ell}^t = [\ell_1^t \ \cdots \ \ell_m^t] \in [0,1]^m$ of which the meta-classifier suffers $L^t = \boldsymbol{\theta}^t \cdot \boldsymbol{\ell}^t$. After the losses have been reviled new weights are calculated by (2.27), where β is an a-priori meta-parameter. When retuning of the weights has finished a new round with incremented t starts. Over time resources are allocated to the advising classifiers who do best.

$$\theta_j^{t+1} = \frac{\theta_j^t \beta^{\ell_j^t}}{\sum_{j \in J} \theta_j^t \beta^{\ell_j^t}} \ \forall j \tag{2.27}$$

Variations of the hedge algorithm that improve its performance exists[10], and more computational intensive spin-offs like AdaBoost[48] have shown promising results[32]. AdaBoost works similarly as the hedge algorithm, but finds its own β through repeated relearning of the classifiers, and gives heavy weights to wrongly classified examples rather than the advising classifiers themselves. However due to the computational strain required for AdaBoost it is not prioritised for this project.

2.3 Related Work

2.3.1 High Frequency Trading

High Frequency Trading (HFT) as mentioned earlier are only similar to this approach in the aspect that they are algorithmic training systems. Aside from this they usually employ very naive investment strategies in order to capture small margins and then repeat this at a very high frequency. What we can learn from these systems is that complexity is not necessary to generate large returns on the market. HFT systems will not be discussed any further.

2.3.2 Time Series Prediction

Many systems aim to predict the price or some other parameter of the stock market. A large difference between these systems is whether they take fundamental data into account. Many predict only based on previous price data, and thus are purely technical. Others add some fundamental data in order to improve the prediction. The latter is however less common. A popular technique for prediction seems to be different regression methods. The traditional one is Autoregressive Moving Averages (ARMA), these are however seldom used as other than benchmarking in modern papers. The most common financial time series predictors use support vector machines (SVM), artificial neural networks (ANN), genetic programming (GP) or other statistical pattern recognition, often combined with fuzzy logic.

Neural Networks

ANNs are the main focus of this project and will thus be discussed at length. Studies suggesting techniques that can supplement ANNs will also be discussed. Frank et al. gives an introduction to time series prediction with neural networks, with important issues like window size and sampling, they conclude that the window size is a crucial determinant of the quality of the prediction [44]. Chan et al. use a neural network with conjugate gradient learning and multiple linear regression (MLR) weight initialization to do financial forecasting[78]. They find it is possible to model stock prices using a three-layer ANN and that MLR speeds up the efficiency of BP. Although many studies uses standard feed-forward networks, it is clear from other studies that specialised ANNs generally outperform these, though often at the cost of added complexity.

Recurrent Neural Networks

According to Saad et al. there are several specific classes of ANNs that performs very good at time series predictions. They did a comparative study of three ANN types specifically apt at time series predictions[92]. These are Recurrent Neural Networks (RNN), Probabilistic Neural Networks (PNN) and Time-Delayed Neural Networks (TDNN). They conclude that RNN are slightly superior, but harder to implement. They use Extended Kalman Filters to train the RNN. Other studies agree that RNNs are well suited for time series predictions [11; 52; 65].

Evolutionary Algorithms

Evolutionary techniques like genetic algorithms (GA) or genetic programming (GP) are often used for financial time series predictions. These are usually used in combination with other techniques, typically ANNs. Hassan, Nath and Kirley creates a fusion model using GA[55]. The ANN is used to generate inputs to the HMM and GA is used to optimise the HMM parameters. They find it to perform equal to the ARIMA model. Kim and Han uses a hybrid model ANN and GA where GA is used to evolve the weights in the network [74]. They conclude that evolution of other parameters parallel with the connection weight worked well, but that their fixed architecture is a limitation. Kim and Shin investigates different hybrids of GA and ANN for their effectiveness of discovering temporal patterns in stock markets[73]. They use adaptive time delay neural networks (ATNN) and time delayed neural networks (TDNN) with GA. They find that the accuracy of the integrated approach is better than standard ATNN and TDNN as well as RNNs. Armano et al. proposes a novel hybrid ANN and GA architecture for stock index forecasting [7]. They create a set of experts that do prediction or classification, these use both

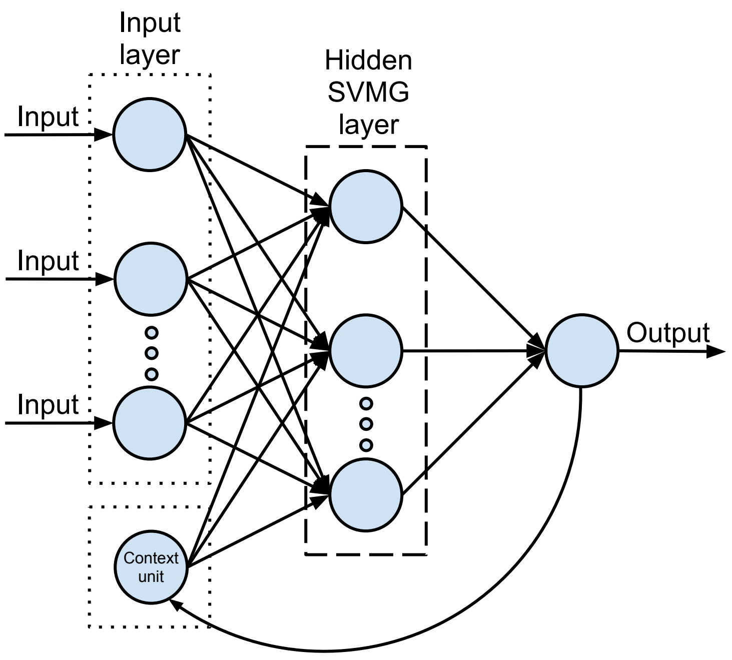


Figure 2.8: This figure shows Pai and Hongs Recurrent SVM with Genetic Algorithms (RSVMG). We can see that their SVM is placed inside a Jordan recurrent network.

GA and ANN techniques. They test it on two stock markets and receive good results.

Support Vector Machines

Support vector machines are starting to dominate the field of time series prediction. Most new systems test these with ANNs as benchmarks, and they generally perform better than standard ANNs. Sapankevych and Sankar [94] do a review of time series prediction using SVM. They point toward many studies showing that SVMs perform better than different ANNs at this task. Several hybrid approaches has also been proposed. Bao et al. propose a self-organizing map (SOM) combined with SVM [13] and find that the hybrid outperform pure SVM. Pai and Hong propose a Recurrent SVM with Genetic Algorithms for parameter search[89]. In their design the SVM act as a layer in their Jordan RNN see figure 2.8. A similar approach is used by Vikramjit et al. [83] where they integrate a RNN and a least squares SVM and achieve good results on text classification. An even more complex model is proposed by Shi and Han [99]. They have a RNN system with a Support Vector Echo-state machine. They receive satisfying results on benchmarks and some real data, but it is not tested on financial data series. Support Vector machines are a relatively novel machine learning approach that shows great promise, and most applications of this new technique is in financial forecasting [94]. Yang and Zhang [114] compares a Support Vector Regression (SVR), a least squares SVM (LS-SVM), BPNN, RBF network and a GRNN for predicting vibration time series. For short time prediction they find that the SVR outperform all, on long term prediction however, the RBF network perform best.

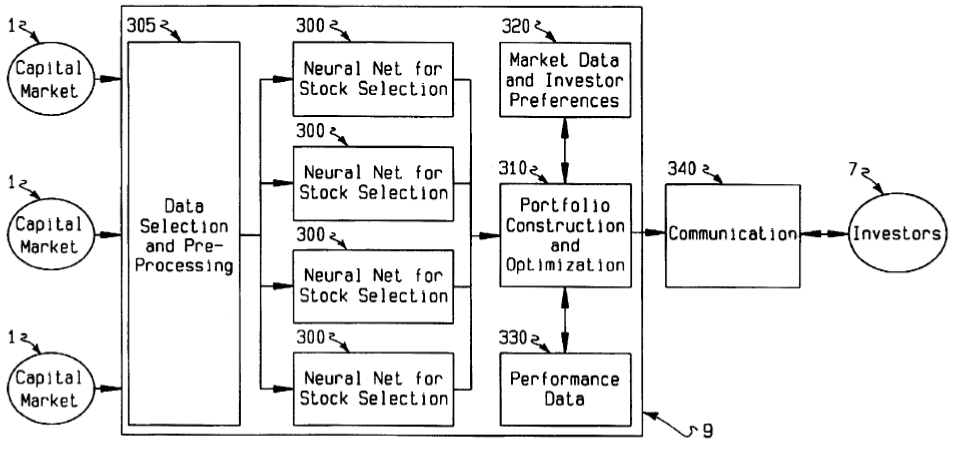

1
Capital Market
1
Capital Market
1
Capital Market
305
Data Selection and Pre-Processing
300
Neural Net for Stock Selection
300
Neural Net for Stock Selection
300
Neural Net for Stock Selection
300
Neural Net for Stock Selection
320
Market Data and Investor Preferences
310
Portfolio Construction and Optimization
330
Performance Data
340
Communication
7
Investors
9

Figure 2.9: This the architecture of Barr's Portfolio optimization using ANNs

Statistical Models

There have been done some promising research into prediction with non-linear statistical models. Li, Liu and Le combines Generalised Regression neural networks (GRNN) with GARCH models and shows it perform better than any of the two alone [76]. Santini and Tettamanzi find that pure genetic programming give very good results, and actually claim that combining this with GARCH, Markov processes and probabilistic learning degrades their performance[93].

Zhang use a hybrid ARIMA and neural network model to forecast time series and finds that this hybrid outperforms any of the models alone[116]. ARMA or ARIMA is used in many other articles, but usually as a benchmark.

2.3.3 Portfolio Optimization Systems

The time series prediction section is concerned with only predicting of single assets or time series. The system is designed to select portfolios and in this section relevant systems for portfolio this use will be presented. Only one system were found using neural networks for actual portfolio selection in the same way as this system plan. This is a patent from 1998, filed by Dean S. Barr [15]. He describes a system quite similar to this system. He has three layers where one is a layer of ANNs that take in pre-processed market data, see figure 2.9. He has his own portfolio construction module, so it is not clear whether the ANNs actually output portfolios like this system will do.

Zhang and Hua use neural network and Markow decision processes to achieve bankruptcy control and dynamic portfolio selection according to Markowitz's mean-variance portfolio selection model[120]. Their results show their system to outperform both pure GA and Linear Programming. Lin and Liu show in their paper that portfolio optimization based on the Markowitz model with minimum transaction lots is NP-Complete, but that a good solution can be achieved using evolutionary heuristics [77]. Zimmerman and Neueneier use neural networks which adapts the Black/Litterman portfolio optimization algorithm. The funds are allocated across securities while simultaneously complying with a set of constraints posted by the user. Gaivoronski, Van Der Wijst and Krylov proposes techniques for optimal selection of portfolios based on a perception of risk and target measures [50]. Their paper further discuss how to optimally rebalance a portfolio given both fixed and proportional transaction cost through a mixed integer LP problem.

2.3.4 Criticism of Sources

Most of the papers found are purely academic. Systems are mostly tested against know benchmarks or some sort of market simulator. None have been found that claims to actually generate a profit from their system. This issue is probably due to the fact that inventors of successful novel approaches would lose much of their advantage by sharing their ideas. Thus they have little or no incentive to publish a practical and working system. There exist many systems out there claiming to employ both ANNs and evolution in their proprietary systems. This is however a field riddled with fraud and it does not seem to be a legitimate claim.

There also seems to be quite a large bias in the literature. First, very few admit failure in their papers. This might be due to the fact that few publish failures, or that researchers continue to tweak or redesign the experiment until it in some way succeeds somewhat. Another aspect that is specific to the financial domain is that many test in “perfect” non-realistic environments. They do not take into account real aspects like brokerage fee, trading delays, volumes and difference in buy and sell prices. This way, many of the systems that in theory generate nice returns would fail in reality.

Chapter 3

Results and Design

3.1 Financial Market Selection

An important design decision is to decide on which financial market the system will operate. There are numerous criteria that affect the suitability of a market, and thus have important implications for the success of the trader.

3.1.1 Market Criteria

Market **liquidity and fungibility** are inherent traits of all financial markets and need not be used as a selection criterion.

Automation complexity refers to the prevalence of exceptions and special cases in the market which are hard to automate. Examples are dividend payments in the stock markets or the start of a new futures contract. This sort of complexity should be avoided in order to improve predictability and reduce system development costs. Automation complexity should not be confused with causal complexity.

Causal complexity refers to the underlying causality between events within a market. A certain degree of causal complexity is required for the algorithmic trader to be successful. An entirely transparent market would tend to be very efficient since it is apparent to all actors what an optimal allocation of resources should be. With added layers of causal complexity it is possible that the market actors have bounded rationality. Bounded rationality is a behavioural trait of economic actors that limits their ability to make rational decisions[85]. On the other hand high causal complexity might make it too hard to extract information from the market.

The **authors' familiarity** with the market is a concern that must be considered due to the risk of trivial pitfalls and resources need to do basic research on the market. This familiarity also greatly influences the quality of input variables chosen for the respective market and how well the implementation is able to account for the deviations mentioned in automation complexity.

There should be a reasonable **research coverage** concerning the market within algorithmic trading. Research cover is seen as an important indicator for interest in the field. This interest is again an indicator on the potential returns that is to be found in the respective market. Additionally previous research is the primary source of information, and the more relevant former studies can be found, the more this system can be improved beyond these. Contrary one can argue that there might be more possibilities to find unexploited weaknesses in less researched markets, but due to the authors lacking knowledge of any such weakness a high degree of research coverage is valued. As mentioned previously, few published studies could also indicate monetary success, and thus lacking incentives to publish, this criteria is thus a trade-off.

The system needs **high quality data which is readily available**. As this is an automated system it is dependent on the availability of the data as a stream. If the system is to be competitive, the latency of this stream must be minimal. Even though the plan is not to compete against HFT, any latency will degrade system performance.

3.1.2 Market Evaluation

Foreign exchange markets have many attractive features for algorithmic trading. There are few exceptions and quirks which indicate low automation complexity. The area is well researched and a high quality data-stream is easily accessed. The underlying causal complexity is traditionally said to emanate from readily available macroeconomics data. This might be undermined by its sensitivity to events external to the market, e.g., political agendas and policy making. Since forex trading is at best zero-sum game[63, pg. 87-90] and has high information asymmetry one can argue that to the determent of outsiders, it is more a loaded gamble than an investment alternative. In a fair zero-sum game between two actors the one with the least cash will lose in the long run, thus the system would need a large amount of capital to stay in this market over a prolonged time period[80].

Stock markets are the quintessential financial market. It has wide research coverage with many data sources of high quality. The risk-return outlook is medium to high, while displaying a certain degree of underlying causal complexity. A fair share of market anomalies have been reported as mentioned in the financial rational part 4.1, supporting an effort to identify arbitrage opportunities. As a complicating matter there are a few event-driven exceptions that affect individual securities, most notably, dividend payments, mergers, bankruptcies and de-listings. Due to the authors' familiarity with the market and the abundance of literature this is found to be a manageable issue.

Bonds have as mentions seldom any meaningful return, and would thus not help the system reach its goal to outperform the market overall. High yielding bonds has a troubled reputation due to recent crises. These are some of the reasons why bonds are not used in this system, it is deemed an altogether poor speculation asset. Bonds are more suited as a safe-haven, and then only the low-yielding ones.

Commodity markets are seldom used in automated trading, one instead uses some derivative derived from the commodity as this is easier to conduct trades with. To trade with the actual commodities is more complex and requires logistics.

Derivatives are commonly used in automated systems. They make it easy to trade based on some asset, without having to go through the hassle of actually buying and selling concrete assets. One can also easily increase or decrease the risk by adjusting the type of bet. Most systems seem to specialize in one derivative like straddles (option) or futures trading. To have a system that trades in several derivatives is unnecessary as one can just adjust the existing one and adds complexity. The authors are not that familiar with derivatives and see no large benefit from trading stocks.

The most relevant candidates have summarized in this table, and ranked them according to the criteria.

	Stocks	**Bonds**	**Commodity Futures**	**Forex**	**Options**
Automation complexity	Medium	Medium	Medium	Low	Medium
Causal complexity	Medium	Low	Medium	Low	Medium
Authors' familiarity	High	Low	Medium	Medium	Low
Research coverage	High	Low	High	High	High
Data quality and availability	High	High	High	High	High
Return/Risk	Medium	Low	Medium	High	High

Stocks have been chosen for the system for the following reasons. First of it is not a strict zero sum game like forex and in the long run one will make a decent return just by following the market. The belief is that overperforming relative to the stock market in general will give a larger return than for instance forex. As mentioned one also needs very heavy capital backing to succeed in forex. Derivatives on stocks or commodities could have been used, but unless one does spread trading, one need more advanced strategies in commodities. Derivatives on stocks can be seen as just a complication on the stocks, and the same return-risk ratio can be achieved on just ordinary stocks. If later options are discovered to have some benefit it will be easy to change to these from stocks. Initially in the system building process stocks are seen as the most legitimate benchmark, easy to implement and easy to access data on.

3.2 Framework Design

The system design builds on a generic online processing framework, see figure 3.1, which encompasses an **input interface**, **processing pipeline** and **output interface**.

3.2.1 Input and Output Interface

The input interface is responsible for fetching data point components from available data sources. The data sources are streams of time series indexed by t. After the input interface retrieves all the individual data point components for time index t, it combines them in to a single "raw" data point $\boldsymbol{x}_1^t$. The data point is then pushed into the processing pipeline, and eventually emerges as a labelling y_t on its way to the output interface which handles communication with a data sink. The output interface can be implemented as simply as a data dump to a human investor, or as complex as a trading adapter that communicates directly to a securities market API for automated trading.

3.2.2 Pipelined Processing Modules

The processing pipeline consists of a sequence of K processing modules which have the domain and co-domain of (3.1) and is described by (3.2) where $i \in I = \{1, 2, ..., K\}$.

$$f_i : \mathbb{R}^{m_i} \to \mathbb{R}^{m_{i+1}} \; \forall i \tag{3.1}$$

$$f_i(\boldsymbol{x}_i^t) = \boldsymbol{x}_{i+1}^t \; \forall i \tag{3.2}$$

In essence the module f_i inputs some real valued data point $\boldsymbol{x}_i^t$ with m_i features, then transforms it into another real valued data point $\boldsymbol{x}_{i+1}^t$ with m_{i+1} features. This repeats for all K modules, then finally outputs the end advice, or label, $\boldsymbol{x}_{I+1}^t = \boldsymbol{y}_t$ for time t. This requires that two consecutive modules have a hand-in-glove relationship such that one module is made to service a specific type of receiving module. Other than this requirement the modules are fully generalised

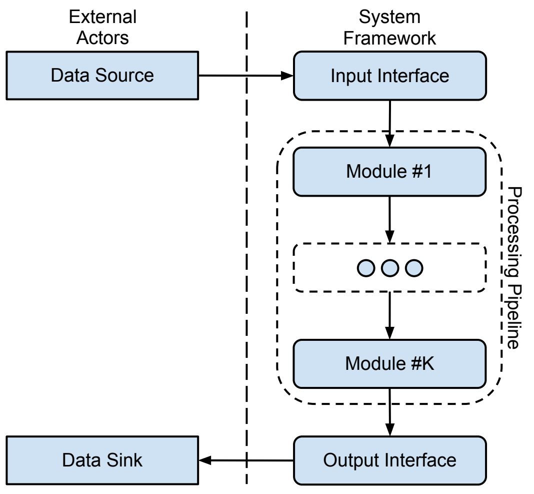


Figure 3.1: Overview of the framework

at the framework level making it possible to adopt many different data representations based on how you implement the processing modules and units.

3.2.3 Processing Units

For simplicity let us drop the module index i and time index t and look at a generic processing module $f(\boldsymbol{x}) = \boldsymbol{y}$ with input $\boldsymbol{x} \in \mathbb{R}^m$ and output $\boldsymbol{y} \in \mathbb{R}^n$. Each processing module contains a collection of L **processing units** of generic construct. They can be implemented as required, and function by performing operations on a feature subset $\boldsymbol{x}_s \subseteq \boldsymbol{x}$ of the processing module's input. The output $\boldsymbol{y}_s$ is of arbitrary dimension. Thus the processing unit can be described by (3.3) where $s \in S = \{1, 2, ..., L\}$.

$$g_s(\boldsymbol{x}_s) = \boldsymbol{y}_s \ \forall s \tag{3.3}$$

With originally m features from the module input and the addition of $m' = \sum_{s=1}^{L} |\boldsymbol{y}_s|$ features from the processing units there can be at most $m + m'$ output features from a module. Here $|\boldsymbol{y}_s|$ represents the dimension of $\boldsymbol{y}_s$. However, there is no requirement for a module to output all of its features and a subset will typically be selected such that the final output has $n \leq m + m'$ features. This implies that some inputs can be passed through unaltered for the next module to process, making it possible to have a strictly sequential data flow. In effect, the processing modules decides what features to transform, discard and leave unaltered for the next module to use. In practice the processing modules must be implemented as more than just a many-to-many function mapping. Each of the processing modules contain an upstream buffer which stores the unprocessed data points it continuously receives from the preceding module. This makes it possible to run the modules asynchronously, aiding scalability due to a desirable pipelining effect.

3.2.4 Learning Scheme

Early stage modules will generally perform simple data preparation tasks, e.g., data mining preprocessing, while later stage modules will typically do more advanced processing, e.g, classification tasks by machine learning generated classifiers. For this to be possible an abstract

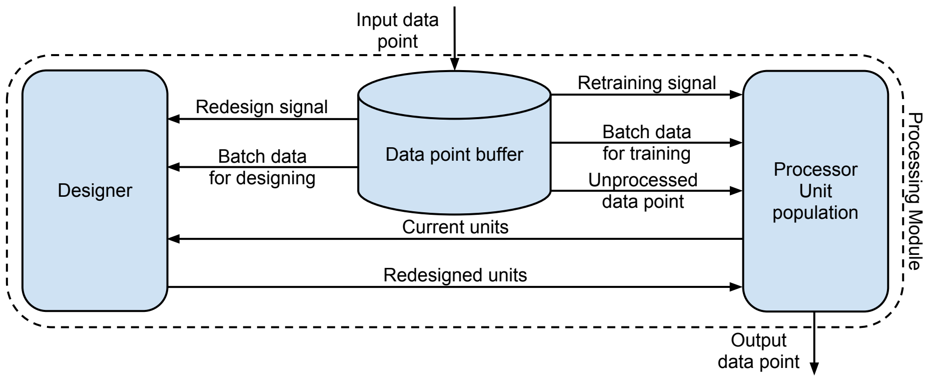


Figure 3.2: The learning scheme illustrated

meta-parameter optimization and sub-parameter learning scheme must be established. The most rudimentary processing units will at most need meta-parameter optimization while more advanced and trainable processing units, later referred to as Agents, will also employ lower level sub-parameter learning. One practical requirement is that the system should be able to perform online learning; it should learn while being deployed and in use. Unfortunately this requirement is not always possible or desirable. A semi-online learning scheme will be constructed that does learning on a schedule, see figure 3.4. Data points that flow into the system will still go through the pipeline and yield an output, however, only after an initial loading phase. When the upstream buffer is filled to capacity with an initial B data points the first design phase of the module is performed. The processor unit's meta-parameters are optimised based on simulated performance on all, or a subset, of the data points in the buffer. From this point on the designer performs a design phase regularly every τ_D time steps. Meta-parameter optimization requires that sub-parameter learning is performed as a subtask. Since sub-parameter learning logically is a less computational intensive ordeal it is possible to perform relearning more often than redesigning. This opens for a regular sub-parameter learning phase every τ_T time steps. Thus, the learning scheme will have to conform to (3.4).

$$1 \leq \tau_T \leq \tau_D \leq B \tag{3.4}$$

3.3 System Design

This section expand on the generic framework establish in the previous section making it applicable to the problem domain. Design decisions are argued for and it is given a more detailed specification of the proposed system implementation. For a more nuanced discussion on design and goal achievement please refer to section 5.2. Since the final system design uses three main processing modules, the focus will be on these and they will be reviewed in turn. After establishing a pipeline design the learning scheme design is elaborated , followed by a discussion of how to best conduct market simulation and trading execution. The system input and output interfaces will not be discussed at depth, as they are practical concerns outside of the project problem domain. For an overview of the implementation see figure 3.3.

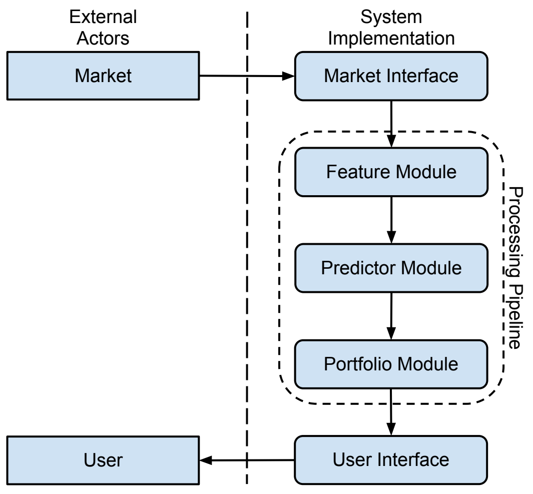


Figure 3.3: High level data flow diagram of system implementation based on the generic framework

3.3.1 Pipeline Design

In general terms, the system contains a pipeline of modules that collectively takes input from the market and output a portfolio recommendation to the user. The **feature model** is responsible for extracting relevant predefined features from raw market data and outputs this to the **predictor module**. Here a set of time series predictions, e.g., regression methods get appended to the feature set and passed on to the **portfolio module**. This final module is responsible for creating candidate portfolios. It then tries to predict which one will be more profitable. Notice that, as described in the framework, how the later modules do more complex computations. This is emphasised by the naming convention attributed to the different module's processing units. The feature module has mere **feature extractors** while the predictor module and portfolio module has **predictor agents** and **portfolio agents** respectively. For a detailed illustration see figure 3.5.

Many agents in both the prediction and portfolio module is subject to supervised learning on two levels. This learning is done both through evolution on the meta-parameters and learning, e.g., backpropagation with artificial neural networks. The details of the learning scheme, including parameter optimisation and learning, will be elaborated on later in this section. A data flow example is illustrated in figure 3.4.

3.3.2 Feature Module

The feature models are responsible for extracting relevant data from the market and presenting it to later modules. It will do simple calculations, but the more complex computations employing learning will be handled by the predictor module. There will be a large amount of different features in this module. These have been presented earlier in the financial data measurement section 2.1.

The input features can be divided into two groups. Those that are linked to a specific stock, and those that are independent. As the system is designed to handle a large amount of different

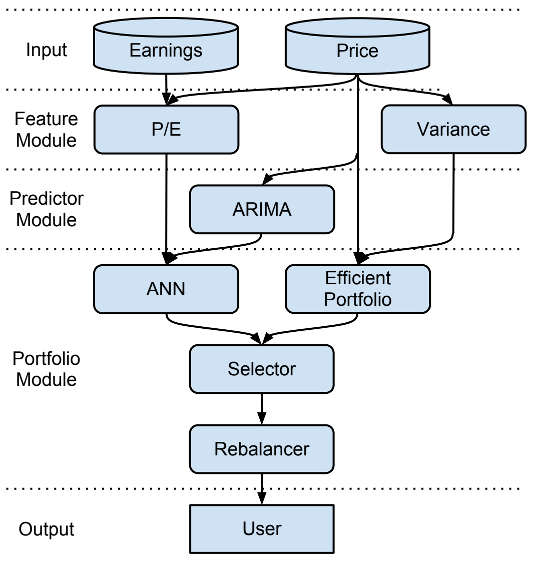


Figure 3.4: Here we see an example of how some data flows through the system pipeline.

stocks, and each of which these will need their own set of features linked to them, there will be a lot of data. Most of this data is only meant for the the main neural network based agent, henceforth known ANN-agent. Most of the other agents have a limited set of specific input requirements. The data will be formatted to fit the components it is sent to, e.g. normalised for the ANNs.

3.3.3 Predictor Module

In order to reduce the load of the agents the system will use a set of more complex predefined features. These are calculated by the raw input features and form a second level of features. All features that require learning is in this category. These advanced features are then sent on together with some unchanged features from the first layer to the portfolio module. Most of these are predictors of different kinds that employ different promising techniques from other studies. As these are not a primary focus they will be implement with existing frameworks when possible.

As shown in the related work section, neural networks have in many cases been shown to perform well at predicting financial time series, see section 2.3. As an addition to the new and experimental use of them as portfolio optimisers they will be uses at something they are known to be good at, namely to predict price changes. The structure of the ANN itself will be quite similar to the one in the portfolio optimizing ANN, the technical details will be elaborated in later in this section. In short it is trained using both evolution and backpropagation. This predictor is designed to run with a fixed set of inputs, but it will be possible to evolve which inputs it is to use. This might be done in a preliminary phase.

Autoregressive Integrated Moving Average

Autoregressive Integrated Moving Average (ARIMA) model are often used in combination with neural networks to predict time series. The ARIMA models are better at linear models, while ANNs generally does non-linear better, thus the combination will be better than any alone.

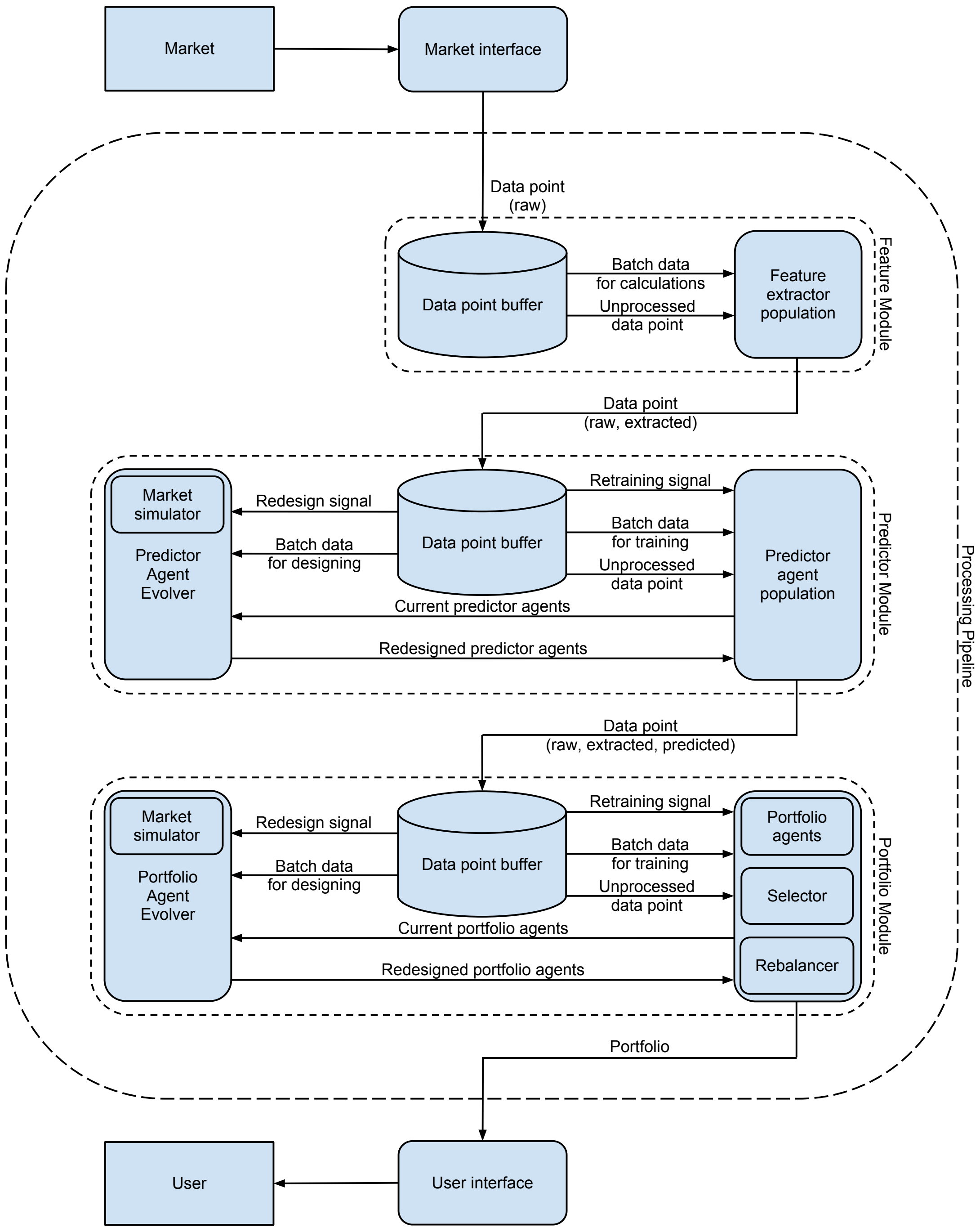


Figure 3.5: This is a lower level data flow diagram illustrating the learning scheme and data point transformations.

ARIMAs are very flexible in what they can represent, but they have a major limitation in that the model is pre-assumed to be linear [116]. The main reasons the system will use ARIMA is to be able to handle linear data and to have a traditional benchmark to measure the other parts of the system against.

Generalised Autoregressive Conditional Heteroskedasticity

Generalised Autoregressive Conditional Heteroskedasticity (GARCH) will be introduced to the system in order to have a volatility prediction better modelled to the nature of financial time series. More specifically GARCH and other ARCH models manage to capture the fact that financial time series volatility have a tendency to occur in bursts [25].

Support Vector Machine

A Support Vector Machine (SVM) will be used in the system to generate a prediction of whether stocks will move up or down. SVMs have shown very promising results in financial forecasting [94]. The strengths of SVM is their ability to accurately predict time series where the underlying models are non-linear, non-stationary and not defined a-priori. There are many successful systems employing a hybrid combination of SVM and RNNs[83; 89; 99].

3.3.4 Portfolio Module

The portfolio module consists of a set of portfolio agent and a portfolio selector. The portfolio agents are responsible for outputting portfolio suggestions to the selector. The selector then output one prediction to the portfolio rebalancer. The rebalancer is responsible for deciding when it is worth rebalancing the portfolio, depending on transaction cost and improvement. This process is illustrated in figure 3.6. The portfolio agents take as input features from the two preceding layers. Which inputs are used is regulated individually for each agent, depending on its type and configuration. All agents of different type will be completely independent. Agents that are trained with evolution will be contained in populations and co-evolved, and thus can be said to have some interaction. The main portfolio agent will be the ANN-agent. Some other and better known portfolio selection strategies have been chosen to supplement it and to benchmark against the ANN-agents. One of the main goals of this paper is to find a design for this agent where it manages to output portfolios that can beat the market. To test whether the agent achieve this goal there will be included some other, better known, agents to compare to. These will also provide some robustness and adaptivity to the system as the selector will be able to shift between different strategies. These other strategies will also be implemented as agents in order to neatly fit in the framework. Some of these will also employ learning, but all to a far lesser degree than the ANN-agent. The portfolio selector will also be able to choose among these agents' predictions.

The ANN-agent

This is the agent that will contain the portfolio optimizing ANN and be the core of the system. The ANN and the mechanisms of learning it will be described in detail later in this section. Its main task is to take various input, spot arbitrage opportunities and generate portfolios that exploit these. It is highly learning dependent and will be trained using both backpropagation and evolution. It will not have a fixed set of inputs as there is no prior way of knowing which inputs will be effective, this is the reason why this aspect will also be put under evolutionary pressure.

This agent will be implemented in several steps. It will start as a feed-forward network. Then be extended to a standard Elman recurrent network and finally to the fully recurrent network. Thus

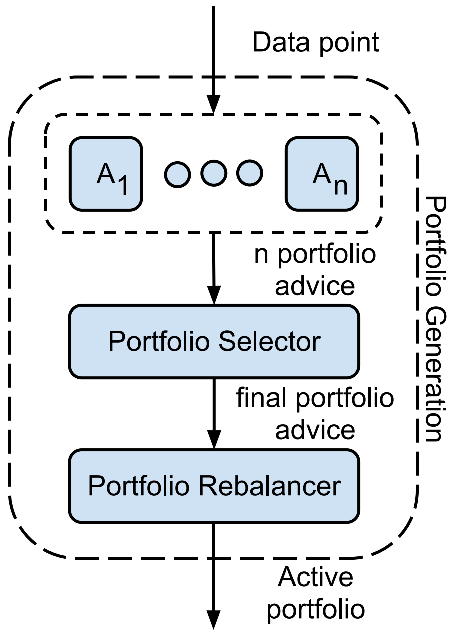


Figure 3.6: An overview of the portfolio module

the performance can measure distance and identify whether the change really is an improvement. It is very plausible that the system will end up using the Elman network as the fully recurrent neural network have not tested on this problem before.

Markowitz Selector

One of the best benchmarks to evaluate the ANN-agent generated portfolios against is a Markowitz portfolio. This is a portfolio that has close to equivalent characteristics as the market portfolio, given the same level of risk, but without the need for as many stocks. This can also be considered as more of a safe-haven compared to many of the other portfolio agents used in the system, though this depends on what risk preference the system is set to. To find the Markowitz portfolio with the minimum transaction lots is a NP-complete problem[77]. However evolution has been shown to be well suitable to find a near optimal Markowitz portfolio. And as it is mainly used as a guide as to what would be a low risk given return portfolio it is not critical to have the absolute minimum transaction lots. Lin and Liu[77] show that evolution is an efficient approach to finding a near optimal portfolio for a given risk or return.

Evolutionary Stock Pickers

This is an approach created in this project. This agent will use only evolution in order to pick a portfolio of n stocks. There are basically two different variations of this agent regarding the distribution of the portfolio. One will just choose a uniform distribution, in order to test only the evolutionary aspect. Another type will choose stocks by evolution and then use those stocks to find a Markowitz portfolio, thus optimising the distribution given the chosen stocks. This is a simple experiment to measure the power of evolution more than a benchmark. This agent will in some aspects be representative for human stock pickers as they also analyse data and through evolution have learned which to pick and which not.

Evolutionary Portfolio Distribution
This agent is quite similar to the ones above. The difference is that this employs evolution for the whole portfolio selection. Thus the genotype will be a vector representing the ratio of each security. This is normalised and a similar threshold as the one above is applied to limit the number of positions. This i thus a purely evolutionary agent, and it will be interesting to see how well it performs.

Portfolio Selector
The portfolio selection is the part of the portfolio module responsible for the output. It takes as input all the prediction of the portfolio agents, and output a new final prediction to the rebalancer. There are many possible implementations possible here. The rationale of having a selector will first be presented, and then the three implementations designed for this system is described.

The intention with this selector is to measure the different agents and somehow always choose the ones that perform well. Thus the system will be more robust and still perform well even if some agents start to malfunction. It may also adapt as some agents may perform better as the environment changes. Some of the approaches combine portfolios. While this might end up with a better portfolio altogether there is a large chance of ruining the portfolio. Portfolios have inherent properties they might lose if modified, thus one cannot just merge two good portfolios and expect to obtain a better one. One big issue is that a merging of two portfolios can potentially double the number of securities without changing the characteristics of the portfolio, thus only resulting in larger transaction cost. An example can be the diversification effect by having reversely correlated assets. If two are merged they may end up with only correlated assets and thus increase systematic risk.

Follow the Leader This portfolio selector will always select one portfolio, which means it only need some metric to choose the best from. Traditional metrics like Sharpe ratio will not do as the Markowitz portfolio is built by maximising this, and they use the MPT assumptions. As the primary goal is return it will base this decision on the historical returns of the portfolio agents. When deciding the length of the time period to calculate the historical return there is a trade-off. A long period will produce a stable, but not very reactive and adaptive solution and vice versa. An idea here is to dynamically adjust this period length according to the predicted volatility of the market, which it can get from the GARCH agent. Thus this selector will dynamically adjust to the temperature in the market.

Evolutionary Weighting The selection process can be evolved like much other in the system. This will have a genotype consisting of a ratio of each of the portfolio agents reflecting how much of their portfolio to include in the aggregated version. To avoid many low positions which cause high transaction costs it will have a minimum threshold on the ratio to be included at all in the aggregated portfolio. As this agent evolves the ideas is that it will find a good way to combine portfolios or choose to not do it at all. The fitness of the phenotype, the portfolio, will be its return over a set time period. The issue with this selector as with all trained ones is that it requires time and data to train.

Hedge Algorithm As with the preceding portfolio selection method, the hedge algorithm allocates weights, or resources, to the various portfolio agents. As time progresses, the algorithm will gradually focus on the most successful agents. Because the hedge algorithm requires that loss conforms to an interval $[0, 1]$ it use a transformed sigmoid function, as in (3.5), to force the absolute return to conform to the specified interval. The meta-parameters β and γ must be set manually or with evolution. High γ makes the loss more sensitive to change in absolute return, while high β makes weight learning happen with more subtle adjustments.

$$\ell = 1 - \frac{1}{1 + e^{-\gamma r}} = \frac{1}{1 + e^{\gamma r}} \tag{3.5}$$

Portfolio Rebalancer

The portfolio selector selects a portfolio for each data point. These might differ much in composition, but the difference in performance might not be as great. The portfolio rebalancer is responsible for deciding when it is favourable to rebalance the portfolio and when it is better to keep the old one. There are several possible approaches to decide this, the one that will be tested for this system will be listed here.

One way is to calculate when the required transaction cost is less than the expected gain in return from rebalancing. This have to been evaluated over a given period, here called t. Then if the expected gain from rebalancing when seen over the period t is greater than the transaction cost it will rebalance.

$$E(v_{\boldsymbol{p}_{\texttt{new}}(T+t)}) - v_{\boldsymbol{p}_{\texttt{new}}(T)} - E(v_{\boldsymbol{p}_{\texttt{old}}(T+t)}) - v_{\boldsymbol{p}_{\texttt{old}}(T)} > \text{Transaction Costs} \tag{3.6}$$

The formula assert that the expected difference in in value over a time t for the new portfolio compared to the old is greater than the transaction costs of changing from the old to the new. The expected value t steps into the future is predicted using one of the predictor agents. Another approach is to compare the new and old portfolio according to some other portfolio metric and then rebalance when this difference reaches a threshold h.

$$f(\boldsymbol{p}_{\texttt{new}}) - f(\boldsymbol{p}_{\texttt{old}}) > h \tag{3.7}$$

Here $f(x)$ is some metric function for the portfolio x in a given time period, and h is the threshold. Both these approaches have an important threshold that needs to be evolved or found in some other way.

The complete system with all of the components discussed in the preceding sections are are modelled in figure 3.7

3.3.5 Learning Scheme

Artificial Neural Network

The system has mainly two different ANN implementations. The main ANN in the system is the one inside of the ANN-agentin the portfolio module. This will be the core of the system that analyses all the input and outputs a portfolio. One of the inputs to this ANN will be another ANN doing price forecasts. This section will show how these two ANNs will be designed.

Both these ANNs will need to be capable of temporal prediction of time series as they are predicting structures in time. Due to their similar capabilities the same basis will be used for both network types, and then let evolution handle the differences. Both the architecture and the connection weights will be evolved. This is inspired by Xin Yao's review on evolving neural networks[115].

Evolution of the architecture and weights is done in the same process to avoid the noise generated when keeping one static while evolving the other [115]. To facilitate this the ANN will be

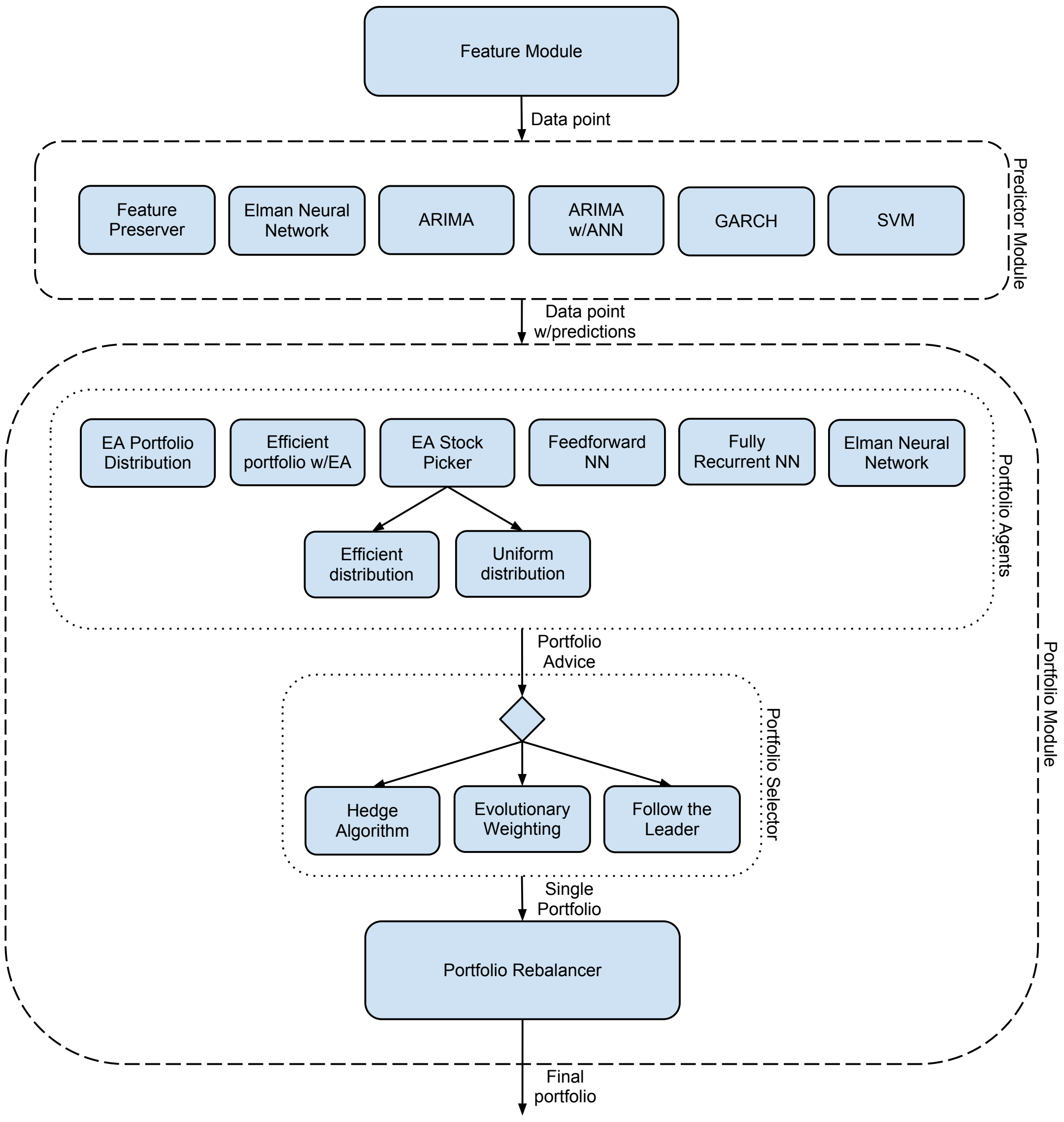


Figure 3.7: Here we see the different specific components mapped onto the architecture of the system. Note that in the portfolio selector, only one of the algorithms will run at any time.

represented as a binary chromosome genotype represented by three matrices, see figure 3.8. One maps input onto the hidden layers, the next represent the mapping between all hidden nodes, and the last represent the mapping from the hidden nodes to the output. The matrices are connection matrices where the number indicates the weight on the connection. By having a full connectivity hidden nodes connection matrix the ANNs will not conform to the traditional hidden layers approach. All the hidden nodes can be thought of comprising the same layer, but these have the ability of creating connection between themselves. These will also be able to form recurrent structures by forming cycles. The network can be forced to be a strict feed forward network by cutting the connection matrix at the diagonal. Both feed-forward and recurrent version of the ANNs will be tested to see whether the expected performance increase and whether the system can effectively train the recurrent ones. In order to achieve temporal predictions from feed forward networks a sliding window is used on the input, the size of this window is another potentially evolvable parameter.

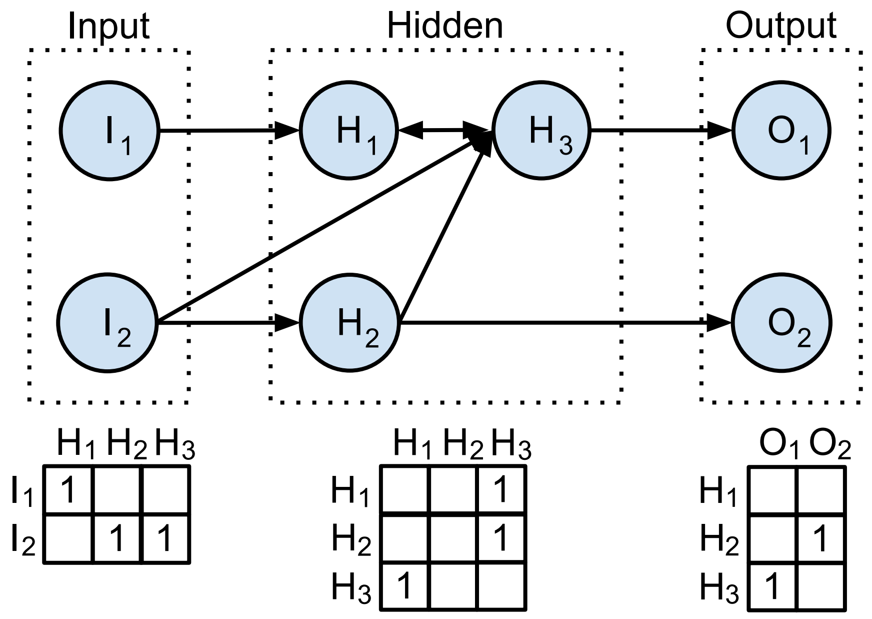
The size of the input layer is set by the number of features used. Due to the enormous amount of potential inputs, the choice of which inputs to use will be put under evolutionary pressure. The size of the output layer will be locked to the number of different securities the agent can choose between. The number of hidden nodes is a non-trivial parameter to set. This is a difficult task as the designers have little expertise in designing ANNs, specifically for this purpose. To ensure enough generalisation ability while avoiding overfitting noise and to high performance demands the number of hidden nodes will be evolved. To penalise overfitting there will be a cost on adding new nodes. Additionally the evolution is run against a validation data set different than the training set to identify when overfitting starts. Overfitting will be further discussed in in the experimental setup section, 3.4. The weight training of the network will in the final implementation be a hybrid between evolution and backpropagation. Xin Yao concluded that this kind of hybrid training in general outperforms both techniques when used separately[115]. The evolution performs well on a broad search, and for avoiding local minima, while backpropagation is effective at the detailed search for an actual optimum. The evolution will first set the initial weights based on the genotype and then backpropagation will be run on both a test and a validation set until the performance on the validation set does not increase.

The criteria in the choice of ANN is its ease of implementation, how well it deals with time and its predictive performance. The implementation will start with a feed forward neural network where it is fed data from several different succeeding points in time in order to give it some notion of the temporal relation of the data. This network will be trained using backpropagation. This is the classical approach to neural networks, with the possible exception of the temporal input organisation. Later this will be extended with evolution and then lastly recurrence.

The implementation will be done using the Encog framework[58]. This is due to its superior performance and ease of use. A rough prototype has been implemented in encog to ensure its feasibility with the intended system. Encog is also licensed with Apache 2.0 making it possible to sell a product containing this framework, as this is not a copyleft license.

Evolutionary Algorithms as Meta-Parameter Optimiser

The primary parameter designer in the system will be evolution. In the system evolution will be used as an on demand parameter search. It will be used one the large parts as the architecture of the ANN and partly on the weights. Further it will also be used to identify good meta-parameters, but these might be locked after identifying a adequate value. In the case of the ANN and its architecture evolution is used to provide an effective search and to introduce adaptivity into the system. As the stock market does large shifts at times, the must be able to quickly adapt.



$$\begin{array}{c|ccc} & H_1 & H_2 & H_3 \\ \hline I_1 & 1 & & \\ I_2 & & 1 & 1 \end{array} \qquad \begin{array}{c|ccc} & H_1 & H_2 & H_3 \\ \hline H_1 & & & 1 \\ H_2 & & & 1 \\ H_3 & 1 & & \end{array} \qquad \begin{array}{c|cc} & O_1 & O_2 \\ \hline H_1 & & \\ H_2 & & 1 \\ H_3 & 1 & \end{array}$$

Figure 3.8: This figure illustrates the mapping between the genotype, the matrices, and the phenotype, the network. Here the three different matrices is illustrated, and how recurrency can be implemented, note that there is a cycle between H1 and H3.

For the system's evolution an existing framework called the Watchmaker evolutionary framework [33] will be used. This framework was chosen among others due to its ease of use and good documentation. The Watchmaker is also licensed under Apache 2.0.

The parameters under evolutionary pressure will be mentioned in their respective parts. They will still be summed up here, and their importance explained. The architecture and weights of the portfolio ANN is the main evolutionary search problem in the system. This is the core of the functionality, and this is a domain where evolution has shown great promise as argued in the ANN section. As the network has a very large amount of potential inputs, the selection of which inputs to use will be put under evolutionary pressure. Aside from these aspects meta-parameters like the learning rate and momentum of the backpropagation algorithm and the window size of feature extraction will be evolved. These meta-parameters might very well be locked after finding a suitable value, as these have less influence on the adaptivity of the system.

Evolutionary Selection of Features

There is a potentially huge feature space. This will create very large ANNs and thus require more training time and data to reach reasonable generalisation ability. The plan is to use some prepossessing to reduce the number of inputs. As Xin Yao points out the problem of finding a near optimal set of inputs can be defined as a search problem[115]. Evolutionary algorithms have been shown to do this search effectively, and given very good results and increased performance [24; 53; 59; 110]. This will easily be implemented through the use of binary chromosomes where "1" indicates use the feature. This can be done as a part of the evolution of the ANN architecture to decrease the noise from having a static architecture.

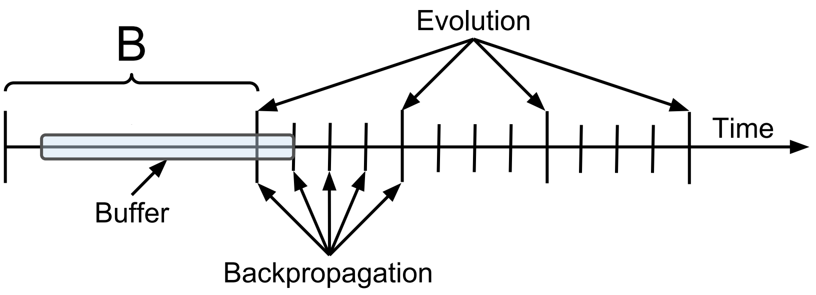


Figure 3.9: This is an illustration of the system time line while running. The buffer is sliding toward right and train each time it hits a vertical line. Note that there can be a higher resolution than the minimal vertical lines illustrated here.

3.3.6 Simulation and Execution

In order to train the system it needs to be fed historical data and receive feedback on its progress. As it is not feasible to do this on live markets this will be handled by the market simulator. The simulator receives the stream of historical price data from the feature module.

Experimental Run

The system will run over two periods, a training period and test period. The training period will be used to train the system, and there will not be done any evaluation at this point. In the test period the system will be run without training and this part will be used to evaluate the system. The system is will first run over a data set of length B designated to training. After this initial training period it will start making predictions. The training buffer will be a sliding window behind, still the length of B, thus gradually dropping old data, see figure 3.9. The system will do more training at certain intervals. There are two levels of relearning, evolution and backpropagation. As mentioned in the framework in section 3.2. Backpropagation will be run more often as it is lighter, and more adept at fine tuning, while evolution is better at broader search. The relearning can be assumed to occur instantly in the evaluation as this can be set out to a parallel process or done during the downtime of the market. By using this sliding approach the system is able to train on data which have already been evaluated on without any issues with overfitting. Each data point will first be evaluated against and then used as training data for the prediction of the next data point. All predictions done and acted on by the rebalancer will result in a continuous position held by the system, and it is the resulting return from this that will be evaluated.

Portfolio Evaluation

One of the largest issues in the simulator is how to evaluate a portfolio given perfect information. This will be the only guidance agents under training will get, and it thus a very critical aspect. Here different approaches will be tested.

One naive solution is to present a portfolio of the one best performing stock, as this will be the best solution given foresight. However this solution does not consider risk. It is still possible that the ANN-agentwill be able to learn to incorporate risk as it would receive a very poor evaluation most times it invested all in one stock. This is one of the approaches to be tested, mainly in order to see how the system will cope with it.

A similar, but slightly more risk aware solution is to choose n stocks with the highest return and then to weight their distribution according to return or according to risk. This risk can be approximated by variance or another risk measure among those mentioned earlier in section 2.1. This could be further extended by measuring the return over several time steps to reduce the noise from very risky stocks.

A more conservative solution is to present a portfolio optimised according to Markowitz's model. Identifying a adequate Markowitz portfolio is still quite computationally heavy as it is NP-complete if one want to minimise the number of transaction lots. The system does however already have the Markowitz selector in the portfolio module, and this could be run one time step ahead in order to supply an evaluation portfolio. The problem with this approach might be that it is to conservative and would due to its nature not be able to exploit anomalies as it is based on modern portfolio theory.

Market Simulation

One of the goals is to make the simulation realistic. The system will to some degree simulate a real stock market environment. This is however a very complex task and it will be simplified. A transaction cost for each buy or sell will be introduced as this is also present in real markets. The transaction cost will be on the form of most real ones with a minimum fee and a percentage thus making the fee (3.8).

$$\text{Transaction Cost} = \max\{k, rt\} \tag{3.8}$$

Where k is the minimum fee, r is the transaction share above this and t is the value of the transaction. In real markets there might be limited supply or demand of a stock, this will be ignored and transaction will be assumed instant. There will also be a **spread** between selling and buying price, these will be assumed equal. As this will make trades cheaper than at real market the transaction cost will be set higher than real market brokerage fees to account for this.

Input Horizon and Resolution

An important question regarding the input is how far back to use input from, and what resolution the data points should be at. Several factors influence this choice. Regarding the historical scope this will affect the dimensions of patterns the system will be able to discover. Too much data will be hard to generalise, while too little will not be applicable outside the local time period. Furthermore the system need a minimum amount of data points to combat overfitting. Resolution affects much the same aspects. Too high of a resolution will contain much more noise and it will be harder to discover long trends. Data at a very low resolution might be unavailable in the stock market. A low resolution will not be as reactive, may miss short trends and require a longer horizon to achieve enough data points. This decision is closely related to the goals of the system. The system will trade intraday, but not at the level of high frequency traders. This resolution is hard to evolve as it is closely linked to the amount of data needed and hard to test independently of the system. This will thus tested on several levels and fixed to the most appropriate one. The hope is to capture intraday patterns, while staying out of the spread trading battles of HFT. The data and time will be inputted with the price features, thus the system will be left to handle exceptions happening around the opening or closing time.

3.3.7 System Limitations

The system design is far from flawless, in this section it will be elaborated on some of the limitations and potential weaknesses the system is predicted to have. One of the main issues expected with the system is the performance. There are many inputs, and many of these need one value for each asset as they are asset dependant. This makes for a large amount of inputs which leads to big and complex ANNs. All this may limit the number of stocks the system can analyse, thus it may lose out on arbitrage opportunities. There is also a limit on the number of input the system can be given, and the reduction in inputs might lead the system to miss out on patterns it could have exploited.

Another issue is the black-box nature of this system. There is no good way of knowing why most agents does the choices they do. This is much of the reason to why evaluation of robustness and adaptivity is introduces so it can be shown statistically that the system still performs stable.

3.4 Experimental Setup and Evaluation Criteria

Because of the black-box nature of system described herein it is important to thoroughly evaluate its performance, and avoid any evaluation pitfalls that invalidate the results. In this effort Cohen and Howe's five stage process is used for evaluating empirical AI systems [30]:

1. Find a task and a view of how to accomplish it
2. Refine the view to a specific method
3. Develop a program implementing the method
4. Design experiments to test the program
5. Run experiments

Ideally this development process should be run as iterative cycles. However, because of the formal structure of this project, and the following thesis, the process have been divided into two phases. This project concerns the design of the system, covering step 1, 2 and 4, while the thesis following this project is dedicated to implementation, covering step 3 and 5.

In this section the planned experiments and performance evaluation criteria on the system as a whole is described. The section is concluded with how overfitting will be detected and avoided.

3.4.1 Evaluation

The system described in this paper is a practical work of engineering. It is designed with the goal to generate a good return on investment. Any evaluation of the AI techniques that the system employs is with that aim in mind. Therefore the evaluation criteria will reflect mostly on how it measures compared to other investment alternatives rather than an AI benchmark problem. All the focus will be on performing on the stock market and thus this is the only data sample the system will use. It will however be tested it on different markets in order to discover whether some might be easier to exploit than others.

Component Evaluation

The system as designed in this paper consists of several largely independent modules consisting of independent agents or other processing modules. The feature module is deterministic and will therefore only be tested for correctness. This can be handled by unit testing[1] and will thus not be an issue. The other modules with agents made to predict or select on a more stochastic basis will have to be qualitatively tested on how well they perform on the task given them, either

[1]A unit test is a simple test of functionality that is meant to be run after each change of the system in order to ensure nothing broke

to predict the next time series value or to select the best portfolio. In order to ensure correct implemented the different modules there will be created a set of deterministic data series to test them on initially. Examples of this is a sinus function, a linear graph etc., to discover if they work at all before the are given"real" financial data where it could be hard to discover whether the fault is in the implementation or just the method cannot handle the data. These trivial test should be so easy and fast that they can be implemented as unit tests to be run continually though the development process to validate that the basic functionality is correct at all times.

The predictions can be tested on historical data, and the error easily calculated. The portfolio agents are harder to evaluate as there are no universally accepted optimal portfolio. The essential parameters are level of risk and return, the issue is how to evaluate the risk. Several portfolio measures have been presented in section 2.1. All of these will be measured as well as return over different time horizons. By running an agent over a very long time horizon the risk will also be reflected in the cumulative return, as it cannot be lucky forever. These tests will be done on real data from stock exchanges like NYSE and OSE. Each agent will also be tested for robustness[2] and adaptivity[3]. To test robustness the system will be exposed to different degrees of artificial noise in the a dataset and the difference in performance will be calculated. The adaptivity will be tested by running it over specific historical dramatic events where the markets changed drastically like financial crisis.

The Selector and the Rebalancer both have deterministic functions and can thus be easily tested. However both have several different strategies that must be evaluated to identify the best, as these will not be dynamically changed during system runs. These can be evaluated by running the system over a long period with many different events like booms and financial crisis and then identify the one that choose wisest. In case of radically different and synergic behaviour several or a combination of the synergetic strategies might be used.

System Evaluation

When the individual components are tested, configured and adjusted accordingly the system can be tested as a whole. The most important criterion is the return, though risk will also be measured in the same way as with the individual portfolio agents. Other relevant criteria are the robustness and adaptivity achieved. The main evaluation of risk-return will be done by running the system over different time horizons with different data resolution and on different stock exchanges (data sources). This will identify the capabilities of the system regarding optimal resolution and time horizon. Adaptivity and robustness will be tested on the system in the same way as the individual components.

The return the system generates will be measured against a representative index. This is the main financial goal. Additionally there will be a set of more scientific goals where the system will be tested against the same subset of stocks as it trades on in order to validate the quality of the technological solution. Further the average annual rate of return will be calculated to have a comparable measure compared to other investments. The different risk measures will be compared with the same measures calculated on a representative set of alternative investments, e.g. stocks, some funds and some indices. Robustness and adaptivity are related to the system itself and will not be compared with anything as there is nothing suitable to compare with. These will however be a measure of how well the system performs.

[2]Robust means how well a system deals with noise, errors and abnormal input

[3]The adaptivity of a system described how well it adapts to new environments, in this case, shifts in the markets

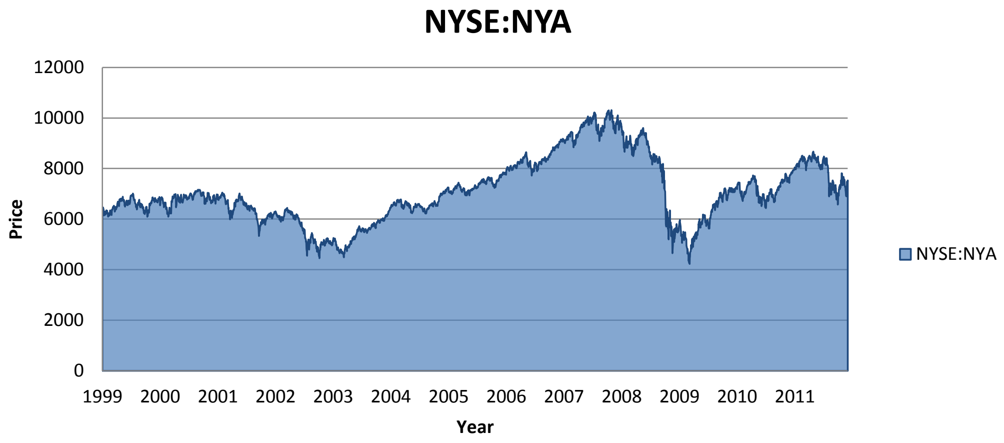


Figure 3.10: This graph shows the development of NYSE Composite Index (NYA) from Jan 1999 to Dec 2011

Evaluation Dataset

The system will be trading stocks on the New York Stock Exchange (NYSE) and Oslo Stock Exchange (OSE). As the system is not designed to have the performance capabilities to process all stocks, a subset of these will be chosen by the system to be as representative for the whole market as possible. Most trading experiments filter away bankrupt companies and other stocks that change their listing during the time period. This introduces a favourable bias as they will avoid investing in companies that goes bankrupt. This will introduce a bias to the system, by using a foresight one does not have when running live. To limit the system's exposure to such companies and to limit the number of stocks it will use a set of a priori constraints that both the evaluation test and live runs will use. This way it does not use foresight, and the evaluation will use the same dataset basis as a real run. As mentioned the goal is to perform better than the market as a whole. Thus it will be natural to evaluate the system against an index representing the market. The system will trade on NYSE and OSE will thus use the NYSE Composite Index (NYA) and Oslo Composite Index(OSEBX). NYSE Composite Index contains all the stocks listed at NYSE. Additionally the system will be compared against a market index made up of its subset of stocks in order to evaluate whether this subset selection affect performance and to what degree. As the system employ much learning overfitting will be a big issue, there have thus been dedicated a section to it.

3.4.2 Overfitting

As mentioned, a big issue in all data mining projects is overfitting. Overfitting is to find patterns in data that only occur from chance and calling it an coherence. When a predictor learns the noise in the data it is overfitting. This is a common issue and one of the most central topics in the field of data mining. The system is particularly prone to this issue as it have several layers of nested learning, and as both neural networks and evolution are highly vulnerable for overfitting. To reduce the chance and degree of overfitting there are several measures that can be taken.

ANN Training ANNs naturally overfit data if they are run too long on the same test data. The ideal is to discover the general patterns that is common to all of the data, not to fit perfectly

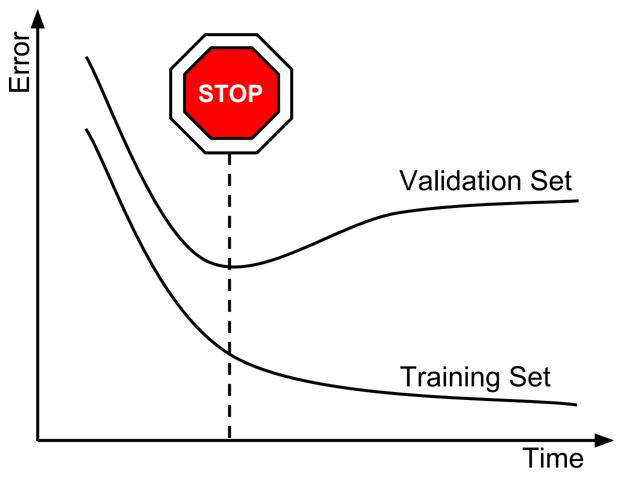


Figure 3.11: Here we see a diagram of the error from the validation set (red) versus the error from the training. At the point marked the training should be stopped to avoid overfitting.

to just this set of training data. To avoid this overfitting the training must be stopped at the right time. This ideal stopping point is the instant when the error on the training set still decreases while the ANN does no better or worse on a general validation set. Thus to find this point the ANN will run against a validation set in addition to the training set. The ANN will however not be adjusted after the validation set as this would contaminate it. When the performance only increase on the training set and not the validation set the training will be stopped, thus eliminating this source of overfitting[102].

Validation One large issue with this approach is where to get the validation data from. The latest 10% could be used, but that would severely delay the responsiveness of the system as it would not be able to learn directly from these. The solution chosen is to randomly select 10% of all data points and use these as validation points. Neural networks will not have large issues with losing a few data points as they will smooth this anyway. Some training data will be lost, but at least they will be spread equally, so much of the larger trend still is intact.

Evolutionary Overfitting The evolutionary training process needs also a stopping point for training. This will be set to an error target, but with an additional limit on the total number of generations for performance reasons. Another approach to discover overfitting in evolution is to measure the homogeneity in the population, if this becomes too homogeneous it indicates that the process is overfitting.

Noise Reduction The data fed into the model will have a certain amount of noise. An important issue is to predict the underlying trend and not the noise of the data. One approach to avoid this noise and thus limit overfitting is to remove it a priori. There are several approaches to removing noise, from simple moving averages to neural networks. Several such methods will be employed and all or an aggregation of these will be presented to the portfolio module. In this way the choice of which noise method is the best is transferred to the system as the input selection is under evolutionary pressure. A potential problem here is that all ”noise” in the price could potentially be exploited by the system, depending on what resolution and scope one have the definition of what is noise will vary. Thus this definition of noise will have to be adapted to the trading resolution of the system.

Amount of Data Points As discussed earlier, most of the system is evolvable. This does not mean all will be evolved at all time. For some of these parameters an adequate value will be found and then fixed. Other parts will only be run at certain intervals of data. The more parameters in an evolutionary search, the larger are the degree of overfitting. This is the main reason to avoid evolving to much of the system at any given time. A rule of thumb is to have at least more than 30 times as many data points as there are weights between hidden nodes in the neural network, when training that. For instance with 50 hidden nodes, and assumed 20% connectivity there should be at least 15000 data points. With four data samples an hour at NYSE, this would mean 577 days of data, roughly equivalent to 2, 3 years. Compared to the scope discussed earlier 2.1 this should not hinder the system in any way.

Diverse Data The other mentioned aspect is to use diverse data. Even though have many similar inputs, e.g. different noise reduced price series etc., it still has several fundamentally different categories of data. An issue here could arise if the evolution of inputs ends up with a relatively homogeneous set of inputs. This is easy to discover and can then be prevented by a fitness reward for having a diverse selection of inputs.

Chapter 4

Discussion

4.1 Financial Rationale

This section will argue for why the system will reach its goals from a financial theoretical perspective.

The purpose of the system is to identify and purchase securities that are underpriced and generate return by selling after the price corrects. According EMH this ought to be impossible as all information should already be reflected in the price, thus foiling any arbitrage opportunities. However, if one looks to behavioural finance they argue that the actors in the market often overreact or make other irrational decisions [70]. They argue that markets may exhibit irrational exuberance emerging from negative or positive feedback loops. This creates opportunities for exploitation[1] of such behaviour and thus generate return by correcting the EMH anomalies. Even Eugene Fama, the father of EMH, has admitted to the fact that there are anomalies in the efficient market model, but retorts that it is only a model, yet a model which, to this day, is the best representation of financial markets we have[38].

According to findings of Bondt and Thaler [21] many actors in financial markets behave irrational. This is one of the foundations of behavioural finance. These results have later been confirmed, and several more similar EMH anomalies have been found. Banz[108] and Reinganum [91] found that firms with low market values of equity was consistently undervalued and vice versa, this is termed the **small firm effect** (SFE). Lamoureux and Sanger [75] showed this same effect on a different market, thus concluding that the effect is independent of market structure and further strengthening the result. This, however, has been challenged by Keim [8] and Brown et al., [26] who found instabilities in SFE. Their results indicate that other factors than posed in SFE can affect stock and the overall market return. This suggests that alternative factors should be incorporated in expected return models. [46].

Other studies have found that the risk-return relationships may be captured by other combinations of company and market specific data. Basu (1983) finds E/P (earnings/price) and firm size both to be constituents of average returns for NYSE/AMEX listed firms[46]. In addition to the psychological arguments against the traditional finance there is the so called "limits to arbitrage" argument.

[1]Here the term "exploitation" is used in a game theoretical sense. That is, changing ones strategy based on another's poorer strategy.

Some argue that underperforming actors will gradually be eliminated as they will lose money through their irrational investments[38]. This is countered by the second foundational block of behavioural finance, the "limits of arbitrage" argument. The argument says that it can be difficult for rational actors to undo dislocations caused by less rational traders[14]. This indicates that there exists latent arbitrage possibilities in a market at any time. The plan is to create a system which is able to exploit these opportunities. This will thus contributing to moving the market towards a more rational and efficient equilibrium, while achieving a profit premium above the average market.

Supported by the behavioural finance arguments the system will use historical data as the basis for trading. It will have none of the cognitive biases which human traders tend to display[70]. Based on the literary research, there seems to be a possibility of identifying arbitrage opportunities. If identified, the system will use those findings to make trading decisions. However, potential issues may arise from two sources:

1. It might not be possible to base trading on historical data. Though, multiple anomalies like SFE and the January effect seems to confirm that it is indeed possible[56; 57; 103].
2. The system may not be able to adequately identify latent arbitrage opportunities.

4.2 Technical Rationale

This section will argue for the rationale behind the technical design mostly by referring to supporting empirical data.

ANNs will be used to forecast the price of each stock. There is much literature in support of this approach. Adya and Collopy found in their review of neural networks applied to forecasting and prediction that out of 22 approved papers, 19 where positive to neural networks, while 3 where negative[5]. They also conclude that much research is poorly validated, and suggest that more effort should be put into mitigating these drawbacks. Zang et al. also agree that ANNs give satisfactory performance in forecasting [118]. Yao and Tan concludes that ANNs are suitable for forecasting of financial time series such as stock indices and foreign exchange rates [66].

ANNs for price forecasting is a heavily researched field that have delivered promising results[44; 60; 78]. According to a comparison done by Saad et al. there are three different ANNs especially apt at predicting price trends[92]. They conclude that Recurrent Neural Networks (RNN) is the most powerful, though not by much, and that the only disadvantage with these is the implementation difficulty. Because of these favourable results, the RNN data structure is chosen to be the default neural network predictor agent.

In order to train these networks a combination of evolutionary algorithm and backpropagation will be employed. Yao shows in his review of evolutionary ANNs (EANN) that evolution has excellent performance on finding good architectures for ANNs. Further more he recommends evolving the weights in the same process[115]. Backpropagation is the quintessential training method for ANNs and a combination of these two approaches in hybrid form has, as mentioned, been successfully validated in many studies[18; 69; 106; 111; 119].

4.3 Time Series Prediction

As mentioned, recurrent neural networks have been chosen for the systems time series predicting ANN. Initially it will start with the standard Elman approach [37], and then extend to a RNN where all hidden nodes can connect to any other. The reason to use Elman and not Jordan is that Elman basically is an improved version of Jordan, and it has been shown that it is possible to train with genetic algorithms[90]. Many researchers agree that these perform very well on time series prediction problems[11; 51; 52; 65]. There are however many other approaches to time series prediction.

As mentioned in the related work, section 2.3, Support Vector Machines (SVM) are very popular in modern systems for prediction problems. Several studies show SVMs outperforming ANNs and other common techniques at this task [89; 101]. It should be noted that the ANNs most studies test against are standard feed-forward networks with backpropagation. A study was done comparing RNN, SVM and ARMA by Thissen et al [104]. They found that SVM and Elman were largely equal in performance on time series predictions and outperformed ARMA. SVM was also found to require much less training data than the Elman network. Due to these satisfactory results an SVM is applied to the prediction module as an agent.

Of late, there has been developed neural networks that show better prediction capabilities on time series. There is a whole family of neural networks that uses radial basis functions (RBF) as activation functions. These are called RBF networks[84]. Generalised regression neural networks (GRNN) are used in some papers for time series prediction [76; 114]. The problem with these and similar approaches is their complexity. The complexity of the system will be increased gradually until it delivers satisfactory result. Other papers have claimed that overly complex systems can often underperform compared to simpler once in real world applications[93], even though they did better in academic settings. More advanced approaches might be applied if the planned ones yield disappointing results. Alternatives include RBF networks or even some of the echo-state systems [94] since they have shown promising results, although they might be more cumbersome to implement.

4.4 Learning Methods

The methods used to train the system are of the same level of importance as the choice of predictors themselves. Both prediction and portfolio agents need training to be effective. Due to their differing structures some need different learning methods. However, some learning methods like evolution is universally applicable to meta-parameters, and may even be used for initial application on learning parameters. For low level parameter optimisation in neural networks variants of backpropagation can be used. For parameter search, few techniques can compete with the ease and effectiveness of evolution. Evolution perform well at broad search, especially where the search space are complex. Evolution is, however, weak at fine tuning. This is where backpropagation shines, thus making the two techniques ideal for combination into a hybrid solution. Many studies have shown that this hybrid approach performs well [18; 69; 106; 111; 119]. It is both well tested and flexible, and thus worthy approach to apply. There are, however, others with several advanced techniques that should be considered. Alternatives to the initial technique will be considered if the hybrid model does not deliver.

One easy adaptation of this method can be to replace backpropagation with conjugate gradient learning. This is essentially the same as backpropagation, but instead of following the gradient it searches along conjugate gradients. This can often produce faster convergence than only following the direction of steepest descent[42]. One interesting approach to training ANNs, especially RNNs, is using **extenden Kalman filters** (EKF). This method adapts the weights of the network pattern-by-pattern accumulating training information in approximate error covariance matrices and providing individually adjusted updates for the network's weights[92]. A use of this method on RNNs are described in Saad et al.[92]. This is however a much more complex approach. The simpler ”evolution-backpropagation” hybrid will be applied first.

4.5 Controlling risk

One of the big questions concerning the performance of the system is whether or not the ANN will learn the risk aspect of portfolio selection. The ANNs are trained from scratch and all abilities these possess is thus a result of feedback during learning. This question then boils down to the examples presented during training. There is a trade-off between encouraging risk consideration and return seeking behaviour. One of the primary goals is to be able to generate a higher return than the market overall. It will not be enough to provide the agents with Markowitz portfolios for training. The problem is that the neural networks would then simply align with the status quo of the market. The agents need to be encouraged to find and exploit anomalies in the market. A hypothesise is that by providing the agents with stocks that have done abnormally well during some time period they will recognise patterns in these abnormal returns. This will however give little indication of the risk taken. Still it is important to note that the system has two levels of training, the evolution, high level learning and the low level backpropagation. The plan is to focus the backpropagation fine tuning toward high return and then evaluate the fitness of the more long term evolutionary training by accumulated gains and losses over a longer time period. This way the evolutionary cycle will be responsible for pruning the population of individuals that take unnecessary risk. The evolution process has the possibility to evaluate over a larger time span and will thus be more suitable to handle the risk aspect, as this is hard to measure at a smaller time periods like the one backpropagation operates on. In order to guarantee a certain risk profile the evolutionary process will be given an acceptable upper and lower bound that its population may operate in. All individuals outside of the bounds will be terminated.

4.6 Portfolio Selection by Neural Networks

As mentioned in section 4.2, Technical Rationale, there is much evidence suggesting the ability of ANNs to predict financial time series. There is, however, less studies suggesting the use of neural network to generate portfolios distributions, or just discrete distributions in general. The only previous case found generating portfolios of this kind is from a patent from 1998 [15]. This problem seems to be a understudied application of neural networks, and it will thus be an important research question. The construction of complete portfolios is a thriving field and even to be able to efficiently construct a portfolio on the same level as the market would be an achievement. A portfolio seen as a whole can have different properties than its parts seen individually. One important aspect is that one can diversify away the non-systematic risk. The ability to select good portfolios is thus an important pursuit that is central to this study.

4.7 Time Resolution and Horizon

An important question for any trader is what time perspective the trades will be done on. For this system questions about total run time, data buffer size, rebalancing frequency and resolution must be considered. Further more, meta-parameter and sub-parameter training frequencies should not be neglected.

The system run time defines the time span the system will run over, this can be from days to decades. This is not to be confused with the buffer size which only reflects how far back in time data is stored. The system could run over ten years with only a buffer size equivalent to one day. What this mean is that a system like this would only considered data at most on day back from the present time when learning an agent's parameters.

The stock market movements shows a fractal property which leads to the same potential gains and losses whatever resolution one might choose. The main difference lies in that there has been shown a tendency for a risk premium, called the equity premium, in long term investment[81]. That is, stock markets seem to increase in value more than alternative markets. This makes long term investments a non-zero-sum, or win-win game. With a short time horizon this premium is negligible, and investment can be view as a zero-sum game.

Most of the existing algorithmic traders present in the markets today are so-called high-frequency traders (HFT). As mentioned in section 2.1 these trade on a microsecond resolution. Most have quite naive strategies and achieve very low margins, but a high return due to the large amount of transactions. In this field there is very hard competition for having the fastest hardware, network and software in order to get best return. As argued earlier this is a zero-sum game, using principles that are not congruent with our design.

Many of the systems data features are not available at very high a resolution. The system should be able to hold positions over a longer time period, and will thus benefit from long term premium of stock markets. This is made possible by rebalancing depending on performance, and not on absolute time. Thus it will dynamically adjust how long each position is held, and the only parameter that needs to be set is the resolution itself. One of the goals of the system is to capture patterns of abnormalities and then react to exploit these. It needs a low enough resolution to be reactive enough to exploit opportunities as well as large enough buffer to capture potential larger patterns, e.g. January effect. The system also needs a large amount of data points in order to avoid overfitting. To the authors' knowledge there is, however, no reason to have more than, at most, one year of historical data in the buffer, as there are few or no anomalies at that scale. The only argument for such amounts of data would be to avoid overfitting, and then it will be better to rather increase the resolution.

Since this is a semi-online trading system with relearning phases, it is important to consider the temporal aspects of the learning schedule. A natural relearning schedule would be to perform meta-parameter optimisation on weekends and sub-parameter relearning on a daily basis. This utilises market downtime periods which are ideal times for the heavy computation required for relearning. Shorter periods could be used with hourly sub-parameter relearning and daily meta-parameter optimisation, although this requires running at least two sets of systems in parallel. One learning while the other is running, and hot-swapping at regular intervals. Alternatives to scheduling is the use of the previously mentioned hedge algorithm which is an online learning algorithm. This reduces the problem to only low level relearning.

Chapter 5

Summary and Conclusion

5.1 Summary

This paper presents the design of an algorithmic portfolio optimization system. The main goals of this system are to be able to outperform the market as well as a capitalization-weighted buy-and-hold strategy on the securities made available to the system.

The system uses a three-layer modular data pipeline to transforms relevant market data into portfolio distribution advice. The modules employ low level processing units that perform data transformation. The feature module performs domain specific feature extraction, which are passed to the predictor module. This module has agents that perform forecasting of relevant time series based on different techniques, and passes these to the portfolio module. The portfolio module contains a set of different agents that selects portfolios. The resulting portfolio distributions created in this module are aggregated or selected amongst for final output. The main part in question is the ANN-agent. This will be implemented using RNN and trained by a hybrid of evolution and BP. The structure of the RNN is very flexible and can easily be transformed into an Elman, Jordan or even a feed-forward network. To capture temporal patterns in the time series, input generation use sliding windows. These can be used directly by neural networks, but are also applicable to autoregressive models, such as ARIMA, GARCH, SVM and other agents in the prediction module. In the portfolio module there are many different agents, all choosing their own portfolio. Aside from the ANN-agentthere will be a Markowitz selector, some evolutionary strategies and a few naive approaches. To give the system some robustness and adaptivity a selector selects the best portfolio given all these advice from the portfolio agents. This selector will be implemented using three different strategies, follow the leader, EA and the Hedge algorithm. In order to reduce transaction costs a rebalancer decides when it is optimal to rebalance based on expected return and transaction costs.

The financial intention behind the system is to be able to discover and exploit anomalies from the efficient market hypothesis. The system is to be built and evaluated in a later project.

5.2 Conclusion

It is found that a modular pipeline with agents will provide the modularity required in the architecture. The number of agents and other components has been limited to a set that can be implemented in less than four months. The modules and components will be individually testable.

A set of neural networks suitable for the task of time series prediction and portfolio selection have been identified. Recurrent and feed-forward neural networks with a hybrid learning scheme with evolution and backpropagation will mainly be used. A module specialised in prediction to aid the portfolio selector agents have been proposed. This module will contain agents using GARCH, ARIMA, SVM and some ANNs. In addition to the ANN portfolio selector several different EA inspired approaches are used. The individual decisions are enhanced by an aggregating selector and a rebalancer takes transaction cost into account when deciding on optimal rebalancing points.

This paper argues that there seem to be anomalies in the EMH in form of arbitrage opportunities; this system aim to exploit these. The system is designed for stock trading, but can easily be extended to trade futures or other derivatives. A large number of inputs to the system have identified, these fall in three main categories. These are technical indicators, fundamental indicators and other contextual data relevant for investment decisions. The evolution is supervised with labels reflecting risk, thus achieving risk consciousness. On the short term learning several strategies have been devised to create label portfolios, most focus on high return. The risk is introduced only in the long run by the evolutionary process as it is easier to determine risk in a longer time frame. The system will be evaluated in a later project on two different stock exchanges and against the composite index of the respective exchanges, additionally risk, robustness and adaptivity experiments have been planned.

5.3 Future Work

As mentioned, this report is just the design of the system, thus the most important future work is to build and evaluate it. This will be done during the spring of 2012. After that evaluation it will be natural to consider extending or adapting the system as required to reach the goal of generating profits above market return. Performance is predicted to be a major issue. Thus to extend the system to be able to run it in parallel on a cluster or similar, and to in a larger degree optimize performance is a natural step further. The system can also be improved on the input side, more advanced fundamental input like interpretation of news can have great effect of the competitiveness of the system.

Bibliography

[1] Oxford English Dictionary A.

[2] The sveriges riksbank prize in economic sciences in memory of alfred nobel 1990, 1990.

[3] Copper, red bull, the world's most informative metal. 2011.

[4] Steven B. Achelis. *Technical Analysis from A to Z.* Probus Publishing, Chicago, 1995.

[5] Monica Adya and Fred Collopy. How effective are neural networks at forecasting and prediction? a review and evaluation. *Journal of Forecasting*, 17(5-6):481–495, 1998.

[6] P.J. Angeline, G.M. Saunders, and J.B. Pollack. An evolutionary algorithm that constructs recurrent neural networks. *Neural Networks, IEEE Transactions on*, 5(1):54 –65, jan 1994.

[7] G. Armano, M. Marchesi, and a. Murru. A hybrid genetic-neural architecture for stock indexes forecasting. *Information Sciences*, 170(1):3–33, Feb 2005.

[8] Donald B. and Keim. Size-related anomalies and stock return seasonality: Further empirical evidence. *Journal of Financial Economics*, 12(1):13 – 32, 1983.

[9] K Reid B Rosenberg. Persuasive evidence of market inefficiency. 1998.

[10] M. Baes and M. Bürgisser. Hedge algorithm and Dual Averaging schemes. *ArXiv e-prints*, December 2011.

[11] Bahman and Kermanshahi. Recurrent neural network for forecasting next 10 years loads of nine japanese utilities. *Neurocomputing*, 23(1-3):125 – 133, 1998.

[12] Arun Bansal, Robert J. Kauffman, and Rob R. Weitz. Comparing the modeling performance of regression and neural networks as data quality varies: a business value approach. *J. Manage. Inf. Syst.*, 10:11–32, June 1993.

[13] Zhejing Bao, Daoying Pi, and Youxian Sun. Short-term load forecasting based on self-organizing map and support vector machine. In Lipo Wang, Ke Chen, and Yew Ong, editors, *Advances in Natural Computation*, volume 3610 of *Lecture Notes in Computer Science*, pages 419–419. Springer Berlin / Heidelberg, 2005.

[14] Nicholas Barberis and Richard Thaler. Chapter 18 a survey of behavioral finance. In M. Harris G.M. Constantinides and R.M. Stulz, editors, *Financial Markets and Asset Pricing*, volume 1, Part B of *Handbook of the Economics of Finance*, pages 1053 – 1128. Elsevier, 2003.

[15] Dean Sherman Barr. Predictive neural network means and method for selecting a portfolio of securities wherein each network has been trained using data relating to a corresponding security, 1998.

[16] S. Basu. Investment performance of common stocks in relation to their price-earnings ratios: A test of the efficient market hypothesis. *The Journal of Finance*, 32(3):pp. 663–682, 1977.

[17] Meredith Beechey, David Gruen, and James Vickery. The efficient market hypothesis: A survey. RBA Research Discussion Papers rdp2000-01, Reserve Bank of Australia, January 2000.

[18] Richard K. Belew, John McInerney, and Nicol N. Schraudolph. Evolving Networks: Using the Genetic Algorithm with Connectionist Learning. In Christopher G. Langton, Charles Taylor, J. Doyne Farmer, and Steen Rasmussen, editors, *Artificial Life II*, volume 10 of *SFI Studies in the Sciences of Complexity: Proceedings*, pages 511–547. Addison-Wesley, Redwood City, CA, 1992.

[19] Sidney Cottle Roger F. Murray Frank E. Block Benjamin Graham, David Le Fevre Dodd. *Graham and Dodd's security analysis.* McGraw-Hill Professional, 1951.

[20] Bruno Biais. High frequency trading, 2011.

[21] Werner F. M. De Bondt and Richard Thaler. Does the stock market overreact? *The Journal of Finance*, 40(3):pp. 793–805, 1985.

[22] R B Boozarjomehry and W Y Svrcek. Automatic design of neural network structures. *Computers & Chemical Engineering*, 25(7-8):1075–1088, 2001.

[23] George Edward Pelham Box and Gwilym Jenkins. *Time Series Analysis, Forecasting and Control.* Holden-Day, Incorporated, 1990.

[24] F.Z. Brill, D.E. Brown, and W.N. Martin. Fast generic selection of features for neural network classifiers. *Neural Networks, IEEE Transactions on*, 3(2):324 –328, mar 1992.

[25] C. Brooks. *Introductory Econometrics for Finance.* Cambridge University Press, 2008.

[26] Philip Brown, Allan W. Kleidon, and Terry A. Marsh. New evidence on the nature of size-related anomalies in stock prices. *Journal of Financial Economics*, 12(1):33 – 56, 1983.

[27] Christopher J.C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2:121–167, 1998. 10.1023/A:1009715923555.

[28] C S Agnes Cheng and Ray McNamara. The valuation accuracy of the price-earnings and price-book benchmark valuation methods. *Review of Quantitative Finance and Accounting*, 15(4):349–370, 2000.

[29] J. Choi. *Technical Indicators.* Jinritamgu Publishing, Seoul, 1995.

[30] P R Cohen and A E Howe. How evaluation guides ai research: The message still counts more than the medium. *AI Magazine*, 9(4):35, 1988.

[31] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20:273–297, 1995. 10.1007/BF00994018.

[32] German Creamer and Yoav Freund. Automated trading with boosting and expert weighting. *Quantitative Finance*, 10(4):401–420, 2010.

[33] Daniel W. Dyer. Watchmaker framework for evolutionary computation, 2006.

[34] Peter D. Easton. PE Ratios, PEG Ratios, and Estimating the Implied Expected Rate of Return on Equity Capital. *SSRN eLibrary*, 2003.

[35] Charles Elkan. Maximum likelihood, logistic regression,and stochastic gradient training, 2011.

[36] Charles Elkan. Predictive analytics and data mining, 2011.

[37] Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179 – 211, 1990.

[38] Eugene Fama. Fama on Market Efficiency in a Volatile Market.

[39] Eugene F. Fama. The behavior of stock-market prices. *The Journal of Business*, 38(1):pp. 34–105, 1965.

[40] Eugene F. Fama. Random walks in stock market prices. *Financial Analysts Journal*, 21(5):pp. 55–59, 1965.

[41] Eugene F. Fama. Efficient capital markets: A review of theory and empirical work. *The Journal of Finance*, 25(2):pp. 383–417, 1970.

[42] R. Fletcher and C. M. Reeves. Function minimization by conjugate gradients. *The Computer Journal*, 7(2):149–154, 1964.

[43] D.B. Fogel. An introduction to simulated evolutionary optimization. *Neural Networks, IEEE Transactions on*, 5(1):3 –14, jan 1994.

[44] R J Frank, N Davey, and S P Hunt. Time series prediction and neural networks. 2001.

[45] Lee N . Price Frank A . Sortino. 1994.

[46] George M Frankfurter and Elton G McGoun. Resistance is futile: the assimilation of behavioral finance. *Journal of Economic Behavior and Organization*, 48(4):375 – 389, 2002.

[47] Craig W. French. Jack Treynor's 'Toward a Theory of Market Value of Risky Assets'. *SSRN eLibrary*, 2002.

[48] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *Proceedings of the Second European Conference on Computational Learning Theory*, pages 23–37, London, UK, 1995. Springer-Verlag.

[49] Alexei A. Gaivoronski and Georg C. Pflug. Value-at-risk in Portfolio Optimization: Properties and Computational Approach. *SSRN eLibrary*, 2004.

[50] Alexei A Gaivoronski, Nico Van Der Wijst, and Sergiy Krylov. Optimal portfolio selection and dynamic benchmark tracking. *European Journal of Operational Research*, 163(1):115–131, May 2005.

[51] C.L. Giles, S. Lawrence, and Ah Chung Tsoi. Rule inference for financial prediction using recurrent neural networks. In *Computational Intelligence for Financial Engineering (CIFEr), 1997., Proceedings of the IEEE/IAFE 1997*, pages 253 –259, mar 1997.

[52] C.L. Giles and Steve Lawrence. Noisy time series prediction using recurrent neural networks and grammatical inference. pages 161–183, 2001.

[53] Z. Guo and R.E. Uhrig. Using genetic algorithms to select inputs for neural networks. In *Combinations of Genetic Algorithms and Neural Networks, 1992., COGANN-92. International Workshop on*, pages 223 –234, jun 1992.

[54] Andrew G Haldane. Patience and finance, 2010.

[55] M Hassan, B Nath, and M Kirley. A fusion model of hmm, ann and ga for stock market forecasting. *Expert Systems with Applications*, 33(1):171–180, Jul 2007.

[56] Mark Haug and Mark Hirschey. The January Effect. *SSRN eLibrary*, 2005.

[57] Robert A. Haugen and Philippe Jorion. The january effect: Still there after all these years. *Financial Analysts Journal*, 52(1):pp. 27–31, 1996.

[58] Jeff Heaton. Encog artificial intelligence framework for java and dotnet, 2008.

[59] Monostori L Hornyak, J. Feature extraction technique for ann-based financial forecasting. *NEURAL NETWORK WORLD*, 7(4-5):543 –552, 1997.

[60] H.Brian Hwarng. Insights into neural-network forecasting of time series corresponding to arma(p,q) structures. *Omega*, 29(3):273–289, Jun 2001.

[61] Stephen J. and Taylor. Forecasting the volatility of currency exchange rates. *International Journal of Forecasting*, 3(1):159 – 170, 1987. ¡ce:title¿Special Issue on Exchange Rate Forecasting¡/ce:title¿.

[62] K. Yeon J. Jun D. Shin H. Kim J. Chang, Y. Jung. *Technical Indicators and Analysis Methods*. Jinritamgu Publishing, Seoul, 1996.

[63] Nick JDouch. *The Economics of Foreign Exchange. Greenwood Press*. Greenwood Press, 1989.

[64] Michael C. Jensen. The performance of mutual funds in the period 1945-1964. *The Journal of Finance*, 23(2):pp. 389–416, 1968.

[65] Won Chul Jhee and Jae Kyu Lee. Performance of neural networks in managerial forecasting. *Int. J. Intell. Syst. Account. Financ. Manage.*, 2:55–71, January 1993.

[66] Chew Lim Tan JingTao Yao. Guidelines for financial forecasting with neural networks. *International Journal of Forecasting*, 2001.

[67] M.I. Jordan. Serial order: A parallel distributed processing approach. *Tech. Rep*, (8604), 1986.

[68] Philippe Jorion. *Value at risk: The new benchmark for controlling market risk*. Irwin Professional Pub. (Chicago).

[69] Hyun jung Kim and Kyung shik Shin. A hybrid approach based on neural networks and genetic algorithms for detecting temporal patterns in stock markets. *Applied Soft Computing*, 7(2):569 – 576, 2007.

[70] Daniel Kahneman and Amos Tversky. Prospect theory: An analysis of decision under risk. *Econometrica*, 47(2):pp. 263–292, 1979.

[71] Con Keating and William F Shadwick. An introduction to omega. *Development*, pages 1–15, 2002.

[72] Con Keating and William F Shadwick. An introduction to omega. *Development*, pages 1–15, 2002.

[73] H Kim and K Shin. A hybrid approach based on neural networks and genetic algorithms for detecting temporal patterns in stock markets. *Applied Soft Computing*, 7(2):569–576, Mar 2007.

[74] Kyoung-jae Kim and Ingoo Han. Genetic algorithms approach to feature discretization in artificial neural networks for the prediction of stock price index. *Expert Systems with Applications*, 19(2):125–132, Aug 2000.

[75] Christopher G. Lamoureux and Gary C. Sanger. Firm size and turn-of-the-year effects in the otc/nasdaq market. *The Journal of Finance*, 44(5):pp. 1219–1245, 1989.

[76] Weimin Li, Jianwei Liu, and Jiajin Le. Using garch-grnn model to forecast financial. *International Journal*, pages 565–574, 2005.

[77] Chang-Chun Lin and Yi-Ting Liu. Genetic algorithms for portfolio selection problems with minimum transaction lots. *European Journal of Operational Research*, 185(1):393–404, Feb 2008.

[78] Chan Man-chung, Wong Chi-cheong, and L A M Chi-chung. Financial time series forecasting by neural network using conjugate gradient learning algorithm and multiple linear regression weight initialization. *Compute*, 2000.

[79] Harry Markowitz. Portfolio selection. *The Journal of Finance*, 7(1):pp. 77–91, 1952.

[80] Peter A. McKay. Scammers Operating on Periphery Of CFTC's Domain Lure Little Guy With Fantastic Promises of Profits. *The Wall Street Journal*, 2005.

[81] Rajnish Mehra and Edward C. Prescott. The equity premium: A puzzle. *Journal of Monetary Economics*, 15(2):145 – 161, 1985.

[82] Geoffrey F. Miller, Peter M. Todd, and Shailesh U. Hegde. Designing neural networks using genetic algorithms. In *Proceedings of the third international conference on Genetic algorithms*, pages 379–384, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc.

[83] Vikramjit Mitra, Chia-Jiu Wang, and Satarupa Banerjee. A neuro-svm model for text classification using latent semantic indexing. In *Neural Networks, 2005. IJCNN '05. Proceedings. 2005 IEEE International Joint Conference on*, volume 1, pages 564 – 569 vol. 1, july-4 aug. 2005.

[84] John Moody and Christian J. Darken. Fast learning in networks of locally-tuned processing units. *Neural Computation*, 1(2):281–294, Jun 1989.

[85] Sendhil Mullainathan and Richard H. Thaler. Behavioral economics. Working Paper 7948, National Bureau of Economic Research, October 2000.

[86] John J. Murphy. *Technical Analysis of the Futures Markets: A Comprehensive Guide to Trading Methods and Applications.* Prentice-Hall, New York, 1986.

[87] Mikhail Myagkov and Charles R. Plott. Exchange economies and loss exposure: Experiments exploring prospect theory and competitive equilibria in market environments. *The American Economic Review*, 87(5):pp. 801–828, 1997.

[88] S. Francis Nicholson. Price ratios in relation to investment results. *Financial Analysts Journal*, 24(1):pp. 105–109, 1968.

[89] P Pai and W Hong. Forecasting regional electricity load based on recurrent support vector machines with genetic algorithms. *Electric Power Systems Research*, 74(3):417–425, Jun 2005.

[90] D.T Pham and D Karaboga. Training elman and jordan networks for system identification using genetic algorithms. *Artificial Intelligence in Engineering*, 13(2):107 – 117, 1999.

[91] Marc R. and Reinganum. Misspecification of capital asset pricing: Empirical anomalies based on earnings' yields and market values. *Journal of Financial Economics*, 9(1):19 – 46, 1981.

[92] E.W. Saad, D.V. Prokhorov, and II Wunsch, D.C. Comparative study of stock trend prediction using time delay, recurrent and probabilistic neural networks. *Neural Networks, IEEE Transactions on*, 9(6):1456 –1470, nov 1998.

[93] Massimo Santini and Andrea Tettamanzi. Genetic programming for financial time series prediction. 2038:361–370, 2001. 10.1007/3-540-45355-5_29.

[94] N. Sapankevych and R. Sankar. Time series prediction using support vector machines: A survey. *Computational Intelligence Magazine, IEEE*, 4(2):24 –38, may 2009.

[95] A. Schroeder. *The Snowball: Warren Buffett and the Business of Life*. Bantam Books. Random House Publishing Group, 2009.

[96] William F. Sharpe. A simplified model for portfolio analysis. *Management Science*, 9(2):pp. 277–293, 1963.

[97] William F. Sharpe. Capital asset prices: A theory of market equilibrium under conditions of risk. *The Journal of Finance*, 19(3):pp. 425–442, 1964.

[98] William F. Sharpe. Mutual fund performance. *The Journal of Business*, 39(1):pp. 119–138, 1966.

[99] Zhiwei Shi and Min Han. Support vector echo-state machine for chaotic time-series prediction. *Neural Networks, IEEE Transactions on*, 18(2):359 –372, march 2007.

[100] J Sietsma and R Dow. Creating artificial neural networks that generalize. *Neural Networks*, 4(1):67–79, 1991.

[101] Francis E H Tay and Lijuan Cao. Application of support vector machines in financial time series forecasting. *Omega*, 29(4):309–317, 2001.

[102] Igor V. Tetko, David J. Livingstone, and Alexander I. Luik. Neural network studies. 1. comparison of overfitting and overtraining. *Journal of Chemical Information and Computer Sciences*, 35(5):826–833, 1995.

[103] Richard H. Thaler. Amomalies: The january effect. *The Journal of Economic Perspectives*, 1(1):pp. 197–201, 1987.

[104] U Thissen, R van Brakel, A.P de Weijer, W.J Melssen, and L.M.C Buydens. Using support vector machines for time series prediction. *Chemometrics and Intelligent Laboratory Systems*, 69(1-2):35 – 49, 2003.

[105] Tim and Bollerslev. Generalized autoregressive conditional heteroskedasticity. *Journal of Econometrics*, 31(3):307 – 327, 1986.

[106] A.P Topchy and O.A Lebedko. Neural network training by means of cooperative evolutionary search. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 389(1-2):240 – 241, 1997. ¡ce:title¿New Computing Techniques in Physics Research V¡/ce:title¿.

[107] John Von Neumann and Oskar Morgenstern. *Theory of Games and Economic Behavior.* Princeton University Press, 1944.

[108] Rolf W. and Banz. The relationship between return and market value of common stocks. *Journal of Financial Economics*, 9(1):3 – 18, 1981.

[109] R. Weatherford. *Philosophical foundations of probability theory.* International library of philosophy. Routledge & K. Paul, 1982.

[110] P.R. Weller, R. Summers, and A.C. Thompson. Using a genetic algorithm to evolve an optimum input set for a predictive neural network. In *Genetic Algorithms in Engineering Systems: Innovations and Applications, 1995. GALESIA. First International Conference on (Conf. Publ. No. 414)*, pages 256 –258, sep 1995.

[111] Werner and Kinnebrock. Accelerating the standard backpropagation method using a genetic approach. *Neurocomputing*, 6(5-6):583 – 588, 1994.

[112] Ronald J. Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural Comput.*, 1:270–280, June 1989.

[113] Michael Wooldridge. *An Introduction to Multiagent Systems.* Wiley, Chichester, UK, 2. edition, 2009.

[114] Junyan Yang and Youyun Zhang. Application research of support vector machines. *Science*, pages 857–864, 2005.

[115] Xin Yao. Evolving artificial neural networks. *Proceedings of the IEEE*, 87(9):1423 –1447, sep 1999.

[116] G Zhang. Time series forecasting using a hybrid arima and neural network model. *Neurocomputing*, 50:159–175, 2003.

[117] G.P. Zhang. Avoiding pitfalls in neural network research. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 37(1):3 –16, jan. 2007.

[118] Guoqiang Zhang, B. Eddy Patuwo, and Michael Y. Hu. Forecasting with artificial neural networks:: The state of the art. *International Journal of Forecasting*, 14(1):35 – 62, 1998.

[119] Ping Zhang, Y. Sankai, and M. Ohta. Hybrid adaptive learning control of nonlinear system. In *American Control Conference, 1995. Proceedings of the*, volume 4, pages 2744 –2748 vol.4, jun 1995.

[120] Yanglan Zhang and Yu Hua. Portfolio optimization for multi-stage capital. *Portfolio The Magazine Of The Fine Arts*, pages 982–987, 2004.

Index