Master of Science Thesis in Industrial Engineering and Management
Department of Management and Engineering, Linköping University
June 2017

Artificial neural networks for financial time series prediction and portfolio optimization

Samuel Björklund
Tobias Uhlin

Credits:	**30HP**
Level:	**A**
Advisor:	**Jörgen Blomvall** Department of Management and Engineering, Linköping University
Examiner:	**Ou Tang** Department of Management and Engineering, Linköping University

ISRN: LIU-IEI-TEK-A--17/02920—SE

LINKÖPING UNIVERSITY

Abstract

Predicting the return of financial times series using traditional technical analysis and widely used economic models such as the capital asset pricing model, has proven to be difficult. Machine learning is a subfield of computer science, a concept that is frequently being used within different domains and recurrently delivers successful results. Artificial neural network is a product from the field of machine learning, a black box model that if properly designed processes data, learns its dynamic and subsequently provides an informative output. The objective of this thesis is to develop artificial neural networks that predict time varying expected return of financial time series with purpose of optimizing portfolio weights.

This thesis begins with a review of prevailing characteristics of financial times series, and existing methods for estimating statistical properties. Background on artificial neural network along with empirical case studies of their suitability within the financial domain is put into a brief discussion vis-a-vis the efficient market hypothesis where potential anomalies is highlighted. Based on the theoretical review, an interdisciplinary approach between machine learning and finance culminated into a model that estimates the expected return of time series on a quarterly basis. To evaluate the use of predictions of future returns in a relevant context a portfolio optimization model, utilizing stochastic programming was built. The financial time series include FX-rates and indices from different asset classes such as equities and commodities.

The results do not show any statistically significant improvement of the expected return estimation compared to CAPM and random walk models. However, our model outperforms a linear regression model for 12 of the 15 time series, which is in line with the performance of previous studies.

Conclusively, no evidence is provided that the proposed model could predict accurately and regardless of the accuracy, the vast portfolio turnover of the model is not congruent with the context for which the model hypothetically would be used.

Acknowledgements

We would like to show our appreciation to our supervisor, Jörgen Blomvall - Thank you for your support during the course of this thesis. Your engagement, comments and ideas have contributed with inspiration and invaluable insights in the fulfillment of this project. We are grateful for your expertise in the numerous courses that you have taught us that are applicable to this thesis.

Also, we would like to express our gratitude to Söderberg & Partners for the opportunity to perform this study. We truly appreciate your friendly and welcoming attitude, a special thanks goes to Meng Chen for supportive advice and the assistance throughout this thesis.

At last, we are grateful for the support from our family and friends. You have all been a source of energy through good and hard times of our studies.

Nomenclature

Abbreviations

Table 1 contains the most commonly used abbreviations in this thesis.

Table 1: Commonly used abbreviations in the thesis and the corresponding phrase

Abbreviation	Phrase
AMH	Adaptive Market Hypothesis
ANN/NN	Artificial Neural Network/Neural Network
ARMA	Autoregressive Moving Average
CAPM	Capital Asset Pricing Model
EMH	Efficient Market Hypothesis
EWMA	Exponentially Weighted Moving Average
FFNN	Feedforward neural network
FX	Foreign Exchange
GARCH	Generalized Autoregressive Conditional Heteroscedasticity
MPT	Modern Portfolio Theory
MSA	Model Selection Algorithm
MSS	Model Selection data Set
OLS	Ordinary least squares linear regression
PCA	Principal Component Analysis
Q-Q	Quantile to Quantile
RW	Random Walk
SP	Stochastic Programming

Machine learning terminology

Table 2 contains the most common machine learning specific terminology used in this thesis. The statistics equivalent terms are provided when possible.

Table 2: Glossary of Machine Learning terminology

Machine learning	**Statistics**
Backpropagation	Repetead chain rule of partial derivatives
Error function	Objective function
Input	Explanatory variables
Model selection data set	In-sample data set
Output	Response variable
Test set	Out-of-sample data set
Training	Optimization
Weights	Regression coefficients

Contents

Chapter 1

Introduction

It is widely accepted that predicting the future with 100 % accuracy is impossible. However, the famous French mathematician Henri Poincaré once said "*It is far better to foresee even without certainty than not to foresee at all*". What if an investor could foresee the statistical properties of financial times series? This thesis will try to improve estimation of expected return versus traditional models by using machine learning with the purpose of generating higher portfolio returns and in the end earn more money.

Modern Portfolio Theory (MPT) is a commonly used investment model that assumes that an investor wants to maximize the expected return given an expected risk level, commonly measured by the standard deviation of the return (Markowitz, 1952). Hence, the model requires not only to estimate the future return, but also to estimate the risk of the portfolio. In MPT an optimal portfolio is defined as a portfolio where the investor is required to raise the risk level in order to achieve higher expected return. (Markowitz, 1952) As such, the investor needs to ately be able to predict both the return of a set of assets and the covariance matrix for the constituents in order to maintain optimal allocation of the assets within the portfolio. Black and Litterman (1992) argue that two fundamental problems with most quantitative portfolio models are to come up with reasonable forecasts of the return, and that small differences in the predicted expected return drastically changes the asset allocations of the portfolio.

The Efficient Market Hypothesis (EMH) suggests that the asset prices on capital markets fully reflect all the available information, and that new information instantly will be incorporated in the price. This implies that an investor cannot exploit any available information to forecast returns. Advocators of the EMH claim that the researchers' and practitioners' attempts to find predictive models are in vain since the only way to generate excess return would be from luck or exposure to riskier assets. (Fama, 1970) However, Grossman and Stiglitz (1980) conclude that if information is costly to obtain the investor will be compensated in accordance to the cost of obtaining the information, later verified by Ippolito (1989).

Many widely recognized economic theories conform to the assumptions and implications of the EMH. One example is the random walk theory that claims that the price of an asset is impossible to predict as it follows a random path (Malkiel, 1973). Another

examples is the Capital Asset Pricing Model (CAPM) that provides a framework for predicting the return of an asset as a function of its risk in relation to the market portfolio. Potential excess return is explained by the risk premium that the market is pricing in to the asset (Sharpe, 1964; Lintner, 1965).

The EMH has been widely disputed. Malkiel (2003) pictures that a growing number of financial economists and statisticians reject the EMH, claiming there is predictive properties in both technical and fundamental information. For example, Lo and MacKinlay (1988) reject EMH by showing that weekly stock prices are predictable in the US, and Lo et al. (2000) propose a non-parametric kernel regression approach to pattern recognition that has predictive properties. Even though Malkiel (2003) expresses his belief in efficient markets in long term, he admits that short-term predictive patterns do exist. One of the implications from the Adaptive Market Hypothesis (AMH) described by Lo (2004) is that predictability in the available information must arise from time to time, else the incentives to gather information leading to price corrections would disappear. Timmermann and Granger (2004) describes that short-term predictive properties do exist in financial markets, but conclude that no stable forecasting patterns do exist, as these will self-destruct when publicly known. An important issue though is that researchers within financial forecasting may be reluctant to publish well performing predictive models, and instead sell them to investment banks (Timmermann and Granger, 2004), something that will possibly delay the process of pattern self-destruction. As simple patterns get known by the public, and consequently looses their predictiveness, new patterns will be increasingly complex, motivating the use of machine learning and pattern recognition models.

An increasingly popular field of financial forecasting is the domain of machine learning, and in particular Artificial Neural Networks (ANN) (Zhang et al., 1998). The worlds largest investment manager, Blackrock Inc., has a firm belief that big data and machine learning approaches will help produce alpha to investors (Blackrock, 2015). ANNs provides desirable properties that some traditional linear and non-linear regression models lack, such as being noise-tolerant (Zhang et al., 1998). Kima et al. (2004) and Szeliga et al. (2003) among others outline neural networks with the property of managing non-stationarity between data. Furthermore, the neural networks are data driven without being restricted by initial assumptions about functional relationships (Qi and Zhang, 2001). According to Hill et al. (1996) the neural networks have been shown to be universal approximators of mathematical functions. Haykin (2009) states that one of the main advantages with ANNs is that they are model-free, which means that the functional relationship will take the form that fits best to the data without explicitly defining the function beforehand. A general functional relationship can be defined as $y = g(\mathbf{x})$, where $\mathbf{x} \in \mathbb{R}^{N\times 1}$ is a set of inputs that maps to a specific output $y \in \mathbb{R}^{1\times 1}$ defined by $g : \mathbb{R}^{N\times 1} \mapsto \mathbb{R}^{1\times 1}$. (Forsling and Neymark, 2011) Finding a relationship between y and $\mathbf{x}$ through a model-free methodology does not require predetermining the mathematical structure of g.

The term Neural Network has it origins in attempts to find mathematical representations of information processing in biological systems (McCulloch, 1943). The concept is derived from the human brain's ability of processing information and learning methodology through the neural system (Agatonovic-Kustrin and Beresford, 1999).

Important assignments of our brain is controlling and steering the human body which demands communication and new learning. These complex tasks are managed by our nervous system which consists of interconnected cells called neurons. They are communicating by firing electrical signals through their existing connections. To enable receiving and sending signals these neurons utilize their dendrites and axons. The dendrites are the receivers which basically are ramifications of the neuron. The axons are the carriers of the signals from the emitting neurons which branches out to establish contact with connected or new neurons to transmit the encoded information. The transmission of information is mediated through the synapse located at the contact point of the emitting neuron's axon and receiving neuron's dendrites. An incoming signal triggers a release of chemical substances which enables the electrical signals to create a voltage difference between the receiving neuron and its environment. If this potential, or more commonly mentioned as activation level, is high enough it generates a spike and the electrical signal is propagated further to other neurons. How the information is encoded in these electrical signals remains being an unsolved question for today's researchers where the strongest hypotheses argues for firing rate or firing time. (Floreano and Mattiussi, 2008)

The human brain has a well developed learning system through its pattern recognition and capability of associating actions through different stimulus e.g, feeling or smelling. McCulloch (1943); Hebb (1949), two of earliest practitioners proposed that classical conditioning is present by neural networks. Rolls and Treves (1998) outline a simple example of associative learning: the sight followed up with the taste of food. Once the stimulis sight and taste of a particular food have been repeated, the human brain have created an association and paired the actions. Further, the taste of food is most commonly eliciting salivation, and once the human brain has created this association between sight and taste the sight of this particular food is now enough to produce salivation while not necessary eating the food. Above example has a strong connection to one of the earliest influences on the behaviorist school of psychology, the famous experiment of Ivan Pavlov, in which he trained a dog to salivate at the sound of a bell, by ringing the bell whenever food was presented (Hagan et al., 2014). With a black box way of explaining this, the stimuli or action could be seen as the input to the box, our brain, which processes the information by creating "hidden" relations between previous inputs and outputs, and then generate a response according to how the box previously associated this specific input with activities.

A fundamental and famous rule within neuroscience describing what happens in the neural system when the human body is adopting and learning is given by Hebb (1949). It was also from this statement a biological explanation of Pavlov's experiment were provided (Hagan et al., 2014). His theory is named the "simple learning rule" which outlines that neurons repeatedly firing together will start a biological growth process. This implies an increase in the effectiveness of the synapses' ability of transmitting electrical signals. An increased effectiveness in transmitting information implies an improved way of communicating between our cells. Consequently, a new ability have been learned or an existing improved. (Floreano and Mattiussi, 2008)

Since ANNs replicate the way the human brain process data it is a quite different way of doing standard statistical analysis (Agatonovic-Kustrin and Beresford, 1999). The

method of artificial neural network have shown successful progress through the latest pharmaceutical research in terms of pattern recognition. ANNs have been applied on complex problems such as detection of cardiac abnormalities from physiological measures and breast cancer from mammograms. A well developed ANN has also shown the capability of exceeding physicians' ability of diagnosing patients. The reason behind its success within the pharmaceutical field is that the data encountered often shows non-linear and complex characteristics that is hard to describe by traditional statistical models. (Guenther, 2001) Its possibility of managing non-linearity without previous knowledge of the characteristics of the data, such as a random unknown sample, gives it a huge potential for further pharmaceutical research (Agatonovic-Kustrin and Beresford, 1999).

Prices of assets on capital markets are complex stochastic processes with characteristics that make the choice of statistical analysis method vital in order to generate trustworthy predictions. Brock and de Lima (1995) conclude that future stock return have a non-linear functional relationship to earlier stock return by carrying out several mathematical tests for linearity. Zhang et al. (1998) state that forecasting time series, such as prices of assets, has long been in the domain of linear statistics but conclude that real world systems often are nonlinear since it is unreasonable that the realization of a given time series is generated by a linear process. The non-linearity of time series has been widely known during the latest decades and several models have been developed, but a considerable downside from these nonlinear model is the required assumption of a functional relationship i.e. they are not being model-free. A major challenge in forecasting unknown time series is the formulation of a model, the functional relationship is often hard to determine upfront as the data set often contains a lot of noise and complex relationships, and a prespecified model will not capture all the features. However, ANNs are unlike to the traditional models not assuming linearity and are opposed to the latter methods by being model-free. (Zhang et al., 1998) The successful results within pharmaceutical research together with is obvious similarities of financial time series incentivize further investigations of its applications in the financial field.

Enke and Thawornwong (2005) proves the prediction capability of ANNs within the financial domain. In the study, three different ANN models outperformed a linear regression model in forecasting performance. Furthermore, the researchers emphasize the relevance of the predictions in portfolio management. Through several metrics a portfolio, with its investments decision based on the ANN predictions, proved excess return, by far, to a portfolio with predictions from linear regression. On top of that, the ANN-portfolio managed to beat a buy-and-hold portfolio during 1992-1999, a strategy associated with EMH.

Freitas et al. (2009) introduce a portfolio optimization model based on the efficient frontier with ANN as predictors of the input parameters. To benchmark their model they evaluate it versus the traditional mean-variance model as well as the corresponding market index. With three different risk profiles they managed to confirm ANNs relevance as prediction methodology through different statistical metrics such as t-test of the excess return. Remarkable was that the high risk profile managed to achieve an accumulated return of 291.9 % and 77.9 % above the benchmarks respectively during

the investigated period of 142 weeks.

Even though several researchers, Enke and Thawornwong (2005); Freitas et al. (2009); Hsu et al. (2016) amongst others, conclude that machine learning models can be used as predictors to generate excess return on the market, the findings from researchers are inconsistent likely depending the non-standardized design process of the ANNs (Zhang et al., 1998). Nevertheless, ANNs are widely used by practitioners in the financial industry including Merrill Lynch & Co., Salomon Brothers, Citibank, The World Bank and more (Widrow et al., 1997).

Prediction and forecasting is ambiguous and fuzzy terminology within the field of statistical research. Trying to define a future return without any flaws or with tiniest error margin is an unreasonable task. A more proper definition of foreseeing return would therefore be to investigate and determine the statistical characteristics of the analyzed data, such as the expected return along with a deviation measure. The aim of this thesis is to investigate and utilize research of ANNs to design a portfolio optimization model that outperforms traditional theories and existing models. In terms of portfolio optimization, estimating statistical characteristics is of highest importance to avoid garbage in, garbage out performance with a well developed optimization model. Highest marginal utility in terms of generating portfolio return can be achieved by estimating expected return. Special focus of this thesis is hence to estimate expected return using artificial neural networks.

1.1 Company background

This master thesis is requested by Söderberg & Partners (S&P) which is a Swedish independent financial counseling company mainly within life insurance, pension and wealth management. The company provides both traditional counseling and discretionary investment management. Founded in 2004, S&P is a major player in the Swedish counseling market and employs roughly 1.200 people. As a part of their customer offer, the company provides recommendations of asset class allocation to their customers quarterly. S&P wants to investigate whether machine learning, and neural network in particular, can be used for predicting financial time series as input parameters to their asset class allocation suggestions. The purpose is to use this asset class allocation[1] as a complement to their existing model of picking assets within an asset class, i.e. fund picking, see Figure 1.1.1 for a visualisation.

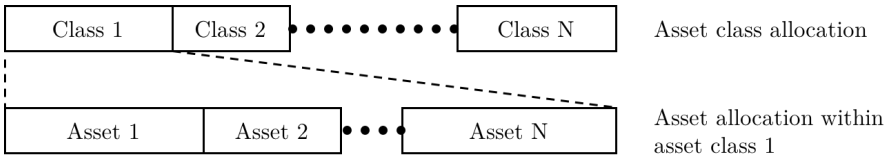

Figure 1.1.1: Visualisation of the asset class allocation and the allocation of assets within an asset class.

[1]Hereafter we will use 'asset' to refer to an 'asset class' approximated by an index

1.2 Objective

To develop artificial neural networks that predict time varying expected return of financial time series with purpose of optimizing portfolio weights.

1.3 Limitations

The model will be limited to forecast and manage a specific number of assets. The assets will be FX-rates and indices of stocks, interest rate and commodity markets corresponding to the markets that S&P provides financial counseling within, see specific assets in Appendix B. The indices are assumed to approximate diversified portfolios of the underlying assets. As such, the portfolio universe correspond to common investment opportunities for a Swedish investor. The time frame for the forecasts and the reallocations is set to a quarter, for which S&P provides regular counselling.

1.4 Disposition

In Chapter 2 the scientific method used for this thesis will be introduced. Chapter 3 will provide an in depth review of the applicable research to the thesis. In Chapter 4 the chosen method will be presented. In Chapter 5 we present the results of our model as well as analyses and evaluation of it. Chapter 6 provides a discussion of the results as well as a proposal of further research in the area. We also conclude our work.

Chapter 2

Scientific method

This chapter contains an overview of the methodology used to answer the objective, i.e. predict time varying expected returns for financial times series in order to optimize portfolio weights. A flow chart of the constituting parts is presented in Figure 2.1.1. In addition, a literature review has to be conducted. In terms of financial times series, the most important properties is the expected returns and covariances of portfolio constituents. Artificial neural networks will be used to determine expected returns. Traditional methods will be used to determine the covariances (i.e. volatility and correlation). This determination will serve as a first phase major of the thesis. Once the financial properties are determined, the portfolio weights can be optimized, which will serve as the second major phase of the thesis. Upon conducting the two major phases, the results will be evaluated, which will later on act as a starting point for concluding the work.

2.1 Literature review

To be able to answer the objective, the research field financial forecasting as well as research of neural networks will be studied, and presented in this thesis. First, applicable economic theory and point estimation in finance will be reviewed. A broader approach to the neural network field will be presented followed by an examination of the usage of neural network within the domain of financial forecasting. Articles, papers and books from a wide range of sources will be used to present a nuanced picture of the research fields. The sources will primarily be accessed via databases that Linköping University's library provides access to. The quality of every source is then evaluated in accordance to which journal it is published in (limited to articles), as well as its number of citations by other papers found in the search engine Google Scholar.

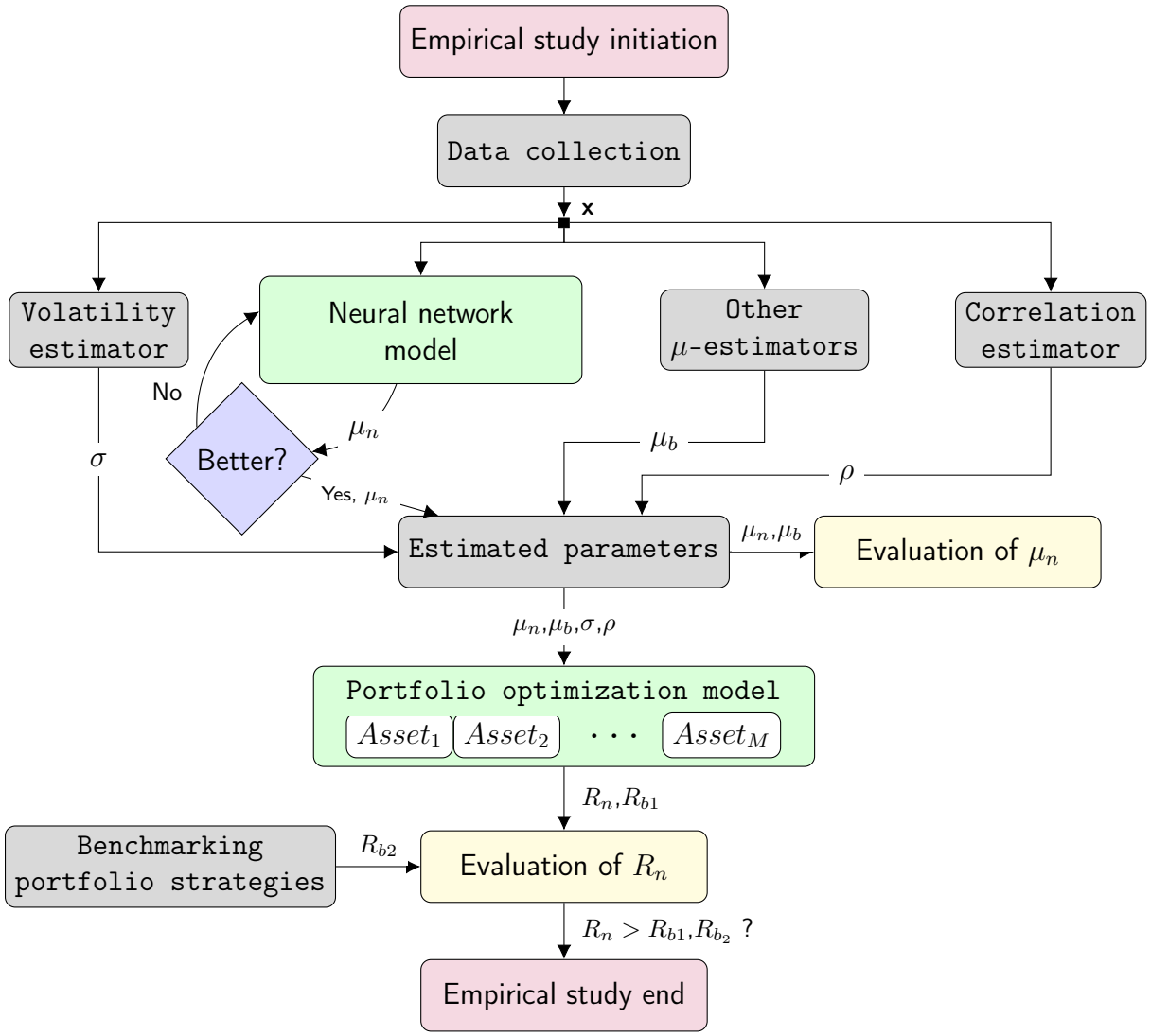

Figure 2.1.1: Flow chart of this thesis where red boxes correspond to start/end point, green boxes as key involving phases, yellow boxes as evaluation parts and last grey boxes as required complementary actions.

2.2 Phase 1: Forecasting using neural networks

Designing a neural network that predicts financial time series require a well thought through process for picking the large number of parameters that needs to be decided upon. Kaastra and Boyd (1996) provides an eight-step procedure for designing neural networks for financial time series forecasting. Based on the vast number of design choices, this eight-step procedure will be the starting point this thesis's methodology design, and is presented below:

Step 1: Variable selection The input parameters used in the neural networks will be technical as well as fundamental. Although one would believe that the more inputs the better, studies show that the predictive power will decrease if too much information is fed to the system as increased noise eventually will confuse the network

(Enke and Thawornwong, 2005). A review of metrics used for prediction in previous research followed by a data selection model will consequently be introduced to find the set of input parameters that maximizes the predictive power of the network.

Step 2: Data collection Data of the financial assets as well as the fundamental data is to be collected via the Bloomberg database.

Step 3: Data preprocessing The data collected from the Bloomberg database will be preprocessed to be suitable for the networks.

Step 4: Training, validation and testing sets The data set will be partitioned into different subsets to fulfill different purposes in the design of the neural networks. One set will be used for training the network to recognize patterns in the data. A second set will be used for the model selection algorithm, and hence generate data that will act as decision basis to choose the configuration that has best predictive power on the specific time series. A third set will be used to provide a final evaluation of the model. It is important to leave this third set truly untouched in order to not bias the design of the networks.

Step 5: Neural network paradigms A common method to describe a neural network is to introduce it as a node network. Gómez-Ramos and Venegas-Martínez (2013) conclude that five types of neural network are used as forecasting models today. In addition to different types of network there are lot of design parameters (hyper-parameters) to choose between as well, which means that a neural network can be constructed in an infinite number of ways. The architecture of the neural network refer to how the network is organized. The network can be divided into three vital sets of layers: input layer, hidden layers and output layer. Choosing the number of layers and the number of nodes within each layer is crucial parameters. Within neural network terminology these are called hyper-parameters. Increasing the number of hidden layers provides the ability to generalize, but in practice a network with only one or perhaps two hidden layer(s) and sufficient number of nodes is enough and have historically shown good performance (Kaastra and Boyd, 1996). Several architectures will be tested for every forecast, and then a model for selecting the number of hidden layers and the number of nodes with the highest performance will be defined.

Further, an important design parameter is the activation function, which specifies how the data is transformed between nodes. The choice of this function will be based on the form of the output data.

Step 6: Neural network training In order for a network to be able to recognize patterns it needs to be presented with observations from the training data set as paired inputs and outputs, called supervised learning. This allows computation of the optimal weights between the neurons. Training will be carried out with an optimization heuristic to find the set of weights that minimizes the error function.

Step 7: Evaluation criteria Once the networks are trained they will be tested on validation data sets. This allows comparison of estimated out-of-sample predictive performance of different architectures. The network that has the best predictive

performance on the training sets for every time series respectively will be assumed to have the best generalization ability. This network will be used for evaluating the neural network technique on the test set.

Step 8: Implementation The actual implementation of the neural network.

2.3 Phase 2: Portfolio optimization

The second stage of the thesis is to define a model for allocating the portfolio weights optimally. There are several possible goals with the optimization, either the expected return (MPT), or the utility of the expected wealth could be maximized where both models involve a certain level of risk preference. In the former case the forecasts from the neural networks and the correlation forecast might directly be used as input to the model. In the latter case scenarios for the return has to be simulated and used as inputs instead, forcing assumptions of the univariate distributions and the dependence between the stochastic variables via a copula. Regardless model, the risk aversion of the investor will impact the optimal portfolio weights, and in the end impact the return of the portfolio.

2.4 Analysis and evaluation

The results from the financial forecasts and the portfolio optimization will be presented, and evaluation will be conducted; one evaluation of the forecasting phase, and one evaluation of the portfolio optimization phase.

In addition to fulfilling the objective of the thesis, it is of interest to evaluate its relevance and thus whether the predictions of the neural networks are good. In order for a prediction to be good, it has to be accurate. We will define accuracy as a model that has less prediction error than well-recognized economic models. Below is a description of how that will be determined.

The forecasting performance of the neural network will be evaluated against traditional methods, e.g. CAPM, for forecasting expected return of the assets. A statistical test will be defined to test whether the neural networks significantly can predict the return better than the reference models. There exists several statistical tests, e.g. Diebold and Mariano (1995), for predictive accuracy with model-free properties which means that the model that generated the forecast does not need to be available. This means that the same test can be used regardless of functional relationship, a desirable property since the specific model will vary per output. The P-value, describing at which significance level the null-hypothesis can be rejected, will be used to decide if a model is significantly better. Regardless of statistical test used, a minimum of 95 % significance will be employed to draw the conclusion that one forecast is better than another.

The asset allocation will be evaluated using widely used single-factor models for portfolio evaluation, such as Sharpe ratio, Treynor Index and Jensen's alpha. The metrics will be compared to the metrics' of a portfolio of equally weighted assets used for the portfolio optimization, as well as a portfolio based on the same portfolio optimization

model, but with another method for estimating the expected return. The evaluation will hence provide basis of discussion regarding the model's ability to generate higher risk-adjusted return and a statistical measure on the significance of the excess return on the market.

The evaluations will be used to prove if the suggested method statistically significant is more accurate respectively perform better than the reference methods. Also, the implications of different method choices and their impact on phase 1 and phase 2 performance respectively will be discussed. Possible improvement areas such as different choice of parameters in the model will be suggested as future research areas. Also, a discussion of what implications our findings has to the efficient market hypothesis will be conducted.

Chapter 3

Theoretical framework

This chapter presents relevant theories regarding the objective of this thesis. The chapter is divided into nine parts, namely: Economic theory, Point estimation, Point estimation in finance, Artificial neural networks paradigm, Data preprocessing, Optimization of neural network parameters, Portfolio optimization, Evaluation, and Empirical tests.

3.1 Economic theory

To understand the domain of expected return estimation in finance, this section aims to introduce relevant economic theory that will aid the neural network design.

3.1.1 Efficient Market Hypothesis (EMH)

The EMH is a theory that stems from Fama (1970), which implies that prices on liquid capital markets fully reflect all available information, and that new information instantly will be incorporated in the price. An implication of EMH is that it is impossible to beat the market, since all available information already is incorporated in the price, and the results by any attempts to do so is subject to the domain of chance. There exists three common forms of the EMH, namely the weak form, semi-strong form and strong form of EMH.

- **Weak form:** The weak form of EMH suggests that the prices on capital markets fully reflect all past prices and volumes, and as such implies that the market cannot be beaten using technical analysis.
- **Semi-strong form:** The semi-strong form of EMH suggests that the prices on capital markets fully reflect all public information. A consequence of this hypothesis is that neither technical nor fundamental analysis can be used to beat the market.
- **Strong form:** The strong form of EMH suggests that prices on capital markets fully reflect all public and private information. This means that consistent excess returns of the market is impossible to achieve consistently, regardless of whether the investor has insider information or not.

(Hull, 2012)

3.1.2 Empirical properties of financial times series

In order to build a neural network model to estimate properties of financial time series, it is important to understand the underlying fundamentals of asset returns. Cont (2001) presents eleven so called stylized facts, which are non-trivial and general statistical properties that are shared between a wide range of instruments and markets, throughout different historical time periods. The eleven stylized facts from Cont (2001) are summarized as follows:

1. **Absence of autocorrelations:** There is often a lack of autocorrelation in asset returns, except for small intra-day time periods ($\sim$ 20 minutes) and weekly and monthly returns
2. **Heavy tails:** The unconditional distribution of returns seems to exhibit heavy tails or Pareto-like tails, however the exact form of the tails is hard to determine
3. **Gain/loss assymetry:** Stock prices and stock index exhibit large drawdowns but not as large movements upwards (FX-rates disregarded)
4. **Aggregational gaussanity:** As the time period over which the return is calculated increases, the distribution looks increasingly as a normal distribution
5. **Intermittency:** At any time scale the asset returns exhibit a high degree of variability, quantified by the presence of irregular bursts in different volatility measures of time series
6. **Volatility clustering:** Volatility exhibits a positive autocorrelation over several days, which implies that high-volatility events are clustered in time
7. **Conditional heavy tails:** The residual time series exhibits heavy tails even after correcting for volatility clustering via e.g. GARCH-models (see Appendix C), although they are less heavy than before clustering correction
8. **Slow decay of autocorrelation in absolute returns:** The autocorrelation of absolute returns decreases as the time lag is increased, sometimes interpreted as a long-range dependence sign
9. **Leverage effect:** Most volatility measures and return of an asset are negatively correlated
10. **Volume/volatility correlation:** Trading volume is correlated with all measures of volatility
11. **Asymmetry in time scales:** Long time scales measures of volatility predict short time scales volatility better than the contrary

3.2 Point estimation

Expected value, variance and correlation are common parameters of common probability density functions. In statistics, these are usually unknown, and consequently

estimated using observations of the random variable of interest. As this thesis aim to estimate expected value using neural networks, we will first examine statistical theory on point estimation.

Let $x_1, x_2, \ldots, x_n$ be observations of the random variables $X_1, X_2, \ldots, X_n$ with density function $f(\mathbf{x}; \boldsymbol{\theta})$ that contains unknown parameters $\boldsymbol{\theta}$. We look for an approximate value of $\boldsymbol{\theta}$, i.e. a point estimation, $\hat{\boldsymbol{\theta}}$, based on $x_1, x_2, \ldots, x_n$. Blom et al. (2005) define a point estimation as a function of observed measured values defined by

$$\hat{\boldsymbol{\theta}} = g(x_1, x_2, \ldots, x_n). \tag{3.2.1}$$

The fix value $\hat{\boldsymbol{\theta}}$ is observations of the estimator vector $\hat{\Theta}$

$$\hat{\boldsymbol{\Theta}} = g(X_1, X_2, \ldots, X_n). \tag{3.2.2}$$

3.2.1 Properties of a point estimator

The distribution for the single random variable $\hat{\Theta}$ determine what values $\hat{\theta}$ can adopt, hence investigating whether the point estimator $\hat{\Theta}$ is biased, consistent and efficient is of certain interest in statistical theory. The estimator is called unbiased if

$$E[\hat{\Theta}] = \theta \tag{3.2.3}$$

and biased if equality does not apply. The bias is defined as $E[\hat{\Theta}] - \theta$. Also the variance of the estimator is of interest, and can be determined as $Var[\hat{\Theta}]$. The variance of the estimator can be estimated using the sample variance $\hat{\sigma}^2$ for n observations of the estimator $\hat{\theta}_i$ as

$$\hat{\sigma}^2 = \frac{1}{n-1} \sum_{i=1}^{n} (\hat{\theta}_i - \bar{\theta})^2 \tag{3.2.4}$$

where $\bar{\theta} = \frac{1}{n} \sum_{i=1}^{n} \hat{\theta}_i$ is the average of the sample of estimations. (Blom et al., 2005)

If large samples are available, asymptotic properties of the estimator can be of interest. An estimator $\hat{\Theta}_n$ for sample size n is said to be consistent if for every $\epsilon > 0$

$$Pr(|\hat{\Theta}_n - \theta| > \epsilon) \to 0, \quad \text{when} \quad n \to \infty. \tag{3.2.5}$$

Blom et al. (2005) formulates the following theorem

Theorem 3.2.1 *If $E[\hat{\Theta}_n] = \theta$ and $Var[\hat{\Theta}_n] \to 0$, when $n \to \infty$, then $\hat{\Theta}_n$ is a consistent estimate of θ.*

If we have two unbiased estimators Θ_1 and Θ_2, then Θ_1 is said to be more efficient than Θ_2 if

$$Var[\Theta_1] < Var[\Theta_2]. \tag{3.2.6}$$

3.2.2 Point estimators

In statistical theory there exist numerous point estimators, with different properties that are beneficial in different settings. Following this, a few of the most common point estimators are described.

Least squares estimation

Consider a sample $x_1, x_2, ..., x_n$ of the random variables $X_1, X_2, ..., X_n$. We assume that $E[X_i] = \mu_i(\theta)$ where $i = 1, 2, \ldots, n$ and $\mu_i(\theta)$ function that is known except for θ. Consequently, $X_i = \mu_i(\theta) + \epsilon_i$ where ϵ_i usually are assumed to be i.i.d. with expected value 0. The squared sum error is now defined as

$$Q(\theta) = \sum_{i=1}^{n} (x_i - \mu_i(\theta))^2. \tag{3.2.7}$$

In the least squares estimation, the $\hat{\theta}$ that minimizes $Q(\theta)$ is the estimate of θ, i.e.

$$\hat{\theta} = \theta^* = \arg\min_{\theta} Q(\theta). \tag{3.2.8}$$

If all the $\mu_i(\theta)$ are identical, we get $\frac{dQ}{d\theta} = -2\mu'(\theta)\sum_{i=1}^{n}(x_i - \mu(\theta))$ which equals to 0 (Q adopts a global minimum) when $\sum_{i=1}^{n}(x_i - \mu(\theta)) = 0 \Leftrightarrow \mu(\theta) = \frac{1}{n}\sum_{i=1}^{n} x_i = \bar{x}$. The value of θ can now be solved out of this relationship, and act as the least square estimate.

(Blom et al., 2005)

Maximum likelihood estimation

A maximum likelihood estimation (MLE) defines the value for unknown parameters that are most likely for a set of samples with a known probability function. Consider a sample $x_1, x_2, ..., x_n$ of the random variables $X_1, X_2, ..., X_n$ with joint probability function $pdf(x_1, x_2, ..., x_n|\boldsymbol{\theta})$ where $\boldsymbol{\theta}$ is a vector of unknown parameters. The likelihood function is defined as $Li(\boldsymbol{\theta}) = pdf(x_1, x_2, ..., x_n|\boldsymbol{\theta})$ and the set of $\boldsymbol{\theta}$ that maximizes the likelihood function is defining MLE, $\boldsymbol{\theta}^*$ as

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} Li(\boldsymbol{\theta}). \tag{3.2.9}$$

Hence, as the estimated vector we have $\hat{\boldsymbol{\theta}} = \boldsymbol{\theta}^*$ In the case of independent and identically distributed sample the likelihood function can be written

$$Li(\boldsymbol{\theta}) = pdf(x_1, x_2, ..., x_n|\boldsymbol{\theta}) = \prod_{i=1}^{n} pdf(x_i|\boldsymbol{\theta}). \tag{3.2.10}$$

For computational purposes, it is often easier to maximize the log-likelihood function in (3.2.11) which has the same likelihood estimator $\boldsymbol{\theta}^*$ as the natural logarithm is a strictly increasing function.

$$\ln Li(\boldsymbol{\theta}) = \ln\big(\prod_{i=1}^{n} pdf(x_i|\boldsymbol{\theta})\big) = \sum_{i=1}^{n} \ln\big(pdf(x_i|\boldsymbol{\theta})\big) \tag{3.2.11}$$

(Blom et al., 2005)

Multiple linear regression

Linear regression is a model that provides a relationship between a number of explanatory variables to a specific response variable. The methodology is to fit a linear relation by using previously observed data points of both the explanatory variables as well as the response variable. The simplest way of doing linear regression is by using one explanatory variable, a model named simple linear regression. For the observations $i = 1, 2, \ldots, n$ the model is denoted

$$y_i = \alpha_0 + \alpha_1 x_i + \epsilon_i \tag{3.2.12}$$

where the parameters α_0 and α_1 describes the intercept and the slope of the line respectively. One could extend simple linear regression to multiple linear regression and use p number of explanatory variables, hence the model is written as

$$y_i = \alpha_0 + \alpha_1 x_{1i} + \alpha_2 x_{2i} + \ldots + \alpha_p x_{pi} + \epsilon_i \tag{3.2.13}$$

where the residual ϵ_i corresponds to the difference between the response observed and the response calculated from the model calculated as $\epsilon_i = \hat{y}_i - y_i$. The vector $\boldsymbol{\alpha}^*$ consisting of the optimal choices of the parameters $\alpha_0, \alpha_1, \ldots, \alpha_n$ are calculated by optimizing

$$\boldsymbol{\alpha}^* = \arg\min_{\boldsymbol{\alpha}} \sum_{i}^{n} \epsilon_i^2. \tag{3.2.14}$$

Thus, the calculated linear regression model can, with known dependent variables, approximate an out of sample response variable as

$$\hat{y} = \alpha_0 + \alpha_1 x_{1i} + \alpha_2 x_{2i} + \ldots + \alpha_p x_{pi}. \tag{3.2.15}$$

(Blom et al., 2005)

3.3 Point estimation in finance

This section describes point estimation in finance, and in particular research on how the expected return can be estimated. Estimations for volatility and correlation can be found in Appendix C.

The future price of a financial asset is by its nature unknown. Consequently, several models have been defined in order to be able to deal with the uncertainty of the prices. The uncertainty is often measured in terms of price changes given a specific time horizon, such as absolute price change, relative price change, or log price change (J.P.Morgan/Reuters, 1996). The absolute price change at time t is defined as

$$S_{\Delta_t} = S_t - S_{t-1} \tag{3.3.1}$$

where S_t refers to actual time series spot price at time t. Relative price changes, referred to as returns, are often preferred though, as these facilitate comparison between assets on different price levels (J.P.Morgan/Reuters, 1996). The percent return is defined as

$$r_t^{perc} = \frac{S_t - S_{t-1}}{S_{t-1}} = \frac{S_t}{S_{t-1}} - 1. \tag{3.3.2}$$

The log price change, log-return (or continuously compounded return) is defined as

$$r_t^{log} = \ln\left(\frac{S_t}{S_{t-1}}\right). \tag{3.3.3}$$

Log-returns are additive across time interval, T, i.e. multi-period returns are computed by a sum of the single-period returns. Percentage returns are additive across assets i, i.e. the return of a portfolio is calculated as a weighted sum of the individual returns. Conclusively, the choice of return metric should be based on the specific application as aggregation convenience differ between the two metrics. Table 3.3.1 provides an overview of the mathematical expressions of the aggregations.

Table 3.3.1: Aggregations of relative returns across time and assets respectively

Type	**Time aggregation**	**Asset aggregation**
Percentage return	$r_{iT}^{perc} = \prod_{t=1}^{T}(1 + r_{it}^{perc}) - 1$	$r_{pt}^{perc} = \sum_{i=1}^{N} w_i r_{it}^{perc}$
Logarithmic return	$r_{iT}^{log} = \sum_{t=1}^{T} r_{it}^{log}$	$r_{pt}^{log} = \ln(\sum_{i=1}^{N} w_i e^{r_{it}^{log}})$

(J.P.Morgan/Reuters, 1996)

Following this, models for estimating the return, the volatility and the correlation will be described. These are important parameters to estimate when building a portfolio optimization model, and since alternative models will be used as a benchmark of the ANN performance a description of the research body on the topic is needed.

3.3.1 Expected return

The expected value of a random variable X is denoted $E[X] = \mu$, and can if the probability density function, $f(x)$, is known be calculated as $\int_{\infty}^{-\infty} xf(x)dx$. In practice, the probability function is rarely known and therefore the expected value has to be estimated. (Blom et al., 2005)

The historic mean of the returns do, according to Amenc and Sourd (2003), provide an unbiased estimate of the expected return[1], for the following period, and can be used as a forecast of the future performance. Accordingly Black and Litterman (1992) claim that the historic mean of returns often is used to predict future returns, but do also stress the fact that the metric has poor predictive performance as future returns are somewhat independent of historic returns. The arithmetic mean of historic returns is defined as

$$\hat{\mu} = \bar{r} = \frac{1}{T}\sum_{i=1}^{T} r_i \tag{3.3.4}$$

and is suitable for log-returns because of its additive property. Using arithmetic mean for percentage returns will overestimate the result, but can be used to estimate the mean percentage return during a period. For percentage returns, the geometric mean is better suited and is defined as

$$\hat{\mu} = \left(\prod_{i=1}^{T}(1 + R_i)\right)^{(\frac{1}{T})} - 1 \tag{3.3.5}$$

giving unbiased mean during the period. (Amenc and Sourd, 2003) The arithmetic and geometric mean provides an equally weighted average of the past returns. An investor that uses historic mean as estimator for future return is faced with the choice of how much past data to use. Amenc and Sourd (2003) points out the fact that the metric will vary vastly depending on which period is chosen, and that the standard practice is to choose three to five years weekly return data. However, the assumption that the past will reproduce itself is rarely true, leading to the development of models that weigh the past returns unequally. Box and Jenkins (1970) popularized the Autoregressive Moving Average (ARMA) model family. The simplest model AR(p) uses a linear function of p past returns to predict the expected return $\hat{\mu}_n$, and is defined as

$$\hat{\mu}_n = c + \sum_{i=1}^{p} a_i r_{n-i} + \epsilon_n \tag{3.3.6}$$

[1]Expected return in this thesis refers to the expected value of the return

where c is a constant, a_i for $i = 1, 2, \ldots, p$ is the weights associated to each past return r_{n-i} for $i = 1, 2, \ldots, p$, and ϵ_n is an error term. The moving average MA(q) model uses a linear function of $q + 1$ weighted random variables ϵ_i, usually assumed to be independent identically distributed (i.i.d.) from a normal distribution with zero mean, i.e. $\epsilon \sim N(0, \sigma^2)$. The model is defined as

$$\hat{\mu}_n = \hat{\mu}_{n-1} + \epsilon_n + \sum_{i=1}^{q} b_i \epsilon_{n-i}. \tag{3.3.7}$$

The ARMA(p,q) model is a combination of the AR(p) and the MA(q) models and is defined as

$$\hat{\mu} = c + \epsilon_n + \sum_{i=1}^{q} a_i r_{n-i} + \sum_{i=1}^{p} b_i \epsilon_{n-i}. \tag{3.3.8}$$

Another model for estimating the return is the Capital Asset Pricing Model (CAPM), which is widely recognized among practitioners and researchers e.g. Sharpe (1964); Lintner (1965). (Amenc and Sourd, 2003) CAPM is, contrary to ARMA models, a development of the EMH. In CAPM the expected return of an asset is related to the expected return of the market portfolio r_M and the riskfree rate r_f. The expected return for an asset is in CAPM defined as

$$\hat{\mu} = r_f + \beta(r_M - r_f). \tag{3.3.9}$$

The market portfolio is the hypothetical portfolio that contains all the world's assets, weighted according to their relative market capitalization. The return of this portfolio is equal to the return of the market as a whole. The riskfree rate is intuitively the theoretical return of an asset with zero risk. The relationship between an assets' return and the return of the market portfolio less the riskfree rate is assumed to be linear. The model is a so called single-factor model, i.e. the expected return can be solely estimated based on the beta value $\boldsymbol{\beta}$ which is defined as

$$\boldsymbol{\beta} = \frac{\sigma_{i,M}}{\sigma_M^2} = \frac{\sigma_i \rho_{i,M}}{\sigma_M}. \tag{3.3.10}$$

Beta can also be estimated with linear regression of past observations. As such, the analyst has to determine how much past data to use in CAPM as well, an disadvantage shared with the mean of historic returns. In practice the return of the market portfolio has to be approximated with the return of a well-diversified portfolio as no market portfolio in the theoretical terms exists. The riskfree rate is often approximated with a three-month treasury bill, as the credit risk is close to zero. Although widely used, CAPM has received criticism from researchers. Roll (1977) argues that CAPM can not be verified as the market portfolio is impossible to observe, and (Fama and French, 1970) invalidates its use in practical applications.

The main assumptions behind CAPM is formulated by Amenc and Sourd (2003) as:

1. *Investors are risk averse and seek to maximise the expected utility of their wealth at the end of the period*
2. *When choosing their portfolios, investors only consider the first two moments of return distribution: the expected return and the variance*
3. *Investors only consider one investment period and that period is the same for all investors*
4. *Investors have a limitless capacity to borrow and lend at the risk-free rate*
5. *Information is accessible cost-free and is available simultaneously to all investors. All investors therefore have the same forecast return, variance and covariance expectations for all assets*
6. *Markets are perfect: there are no taxes and no transaction costs. All assets are traded and are infinitely divisible*

Several extensions to CAPM have been proposed that incorporate additional factors, so called multi-factor models. Example of those is the Fama French three factor model that in addition to the market return adds a book-to-market[2] factor and a size[3] factor. The Carhart four factor model is an extension of the Fama French three factor model, and adds a momentum term. (Amenc and Sourd, 2003) A recent paper by Fama and French (2015) considers the five factors market, book-to-market, size, profitability[4] and investment[5].

The CAPM is not valid on foreign exchange rates. Instead several other theoretical models have been proposed throughout history, both based on economic theory, such as the uncovered interest rate parity, purchasing power parity, as well as time series analysis, such as AR. Meese and Rogoff (1983) compare the performance of several structural and times series models[6] for predicting the FX rate, and finds that none of these models do outperform a random walk model. This is in line with the findings of Mussa (1986), who states that "*The natural logarithm of the spot exchange rate follows approximately a random walk*", as well as the more recent study by Cheung et al. (2005) that draw the conclusion that none of the evaluated models[7] outperform random walk on the metric MSE, but some of them do statistically outperform the random walk model on a direction-based prediction. They also conclude that a model, specifications and currencies that work well in one period might not do in another period.

[2]Book value of the equity in relation to the market value of the equity

[3]Market capitalization

[4]Operating profitability

[5]Total assets growth

[6]Models evaluated: Forward rate, Frenkel-Bilson, Dornbusch-Frankel, Hooper-Morton, Univariate autoregression, vector autoregression. Time horizon 1-12 months. FX-rates: USD/GBP, USD/DM, USD/JPY.

[7]Models evaluated: Interest rate parity, Productivity based models, composite specification incorporating the real interest differential, portfolio balance and nontradables price channels, purchasing power parity and the Dornbusch-Frankel sticky price monetary model. Time horizon: 1, 4, 20 quarters. FX-rates: USD/CAD, USD/GBP, USD/DM, USD/JPY.

3.4 Artificial neural networks paradigm

In this thesis the paradigm of an artificial neural network refers to the different characteristics one could set when determining a neural network model. Before introducing the different paradigms of a network one must determine if an ANN is the right approach to solve the problem encountered. If so, there is of highest interest to use the right network that suits the problem. Hence, we will introduce the statistical problem definition of the broader field, machine learning. We will define the notions learning rule and learning task i.e. what approach to use to make the algorithm learn and respectively the problems that is encountered within machine learning, and further on present some tasks where a neural network approach is suitable.

3.4.1 Machine Learning

Neural network is an approach within the field of machine learning, which basically and with a short description means, constructing algorithms that solves problems and make predictions. This is done by learning the characteristics and testing the algorithm on sample data to enable entering static equilibrium to a specific output given a new specific data set of inputs. (Bishop, 2006)

Learning rule

The learning methodology, i.e. the learning rule, of the algorithm is classified into supervised and unsupervised learning. Supervised learning is a method that presents the algorithm with both input and the subsequent output data. The input data targets the specific output data and the algorithm learns the mapping procedure. Whereas in unsupervised learning there is no corresponding target output of the input and the model is expected to determine them by itself, a sequence of input is provided but no corresponding output values. (Haykin, 2009; Bishop, 2006)

General learning tasks

The task or the problem that machine learning can encounter are many, but there are two more commonly encountered where a machine learning approach is used. Those are directly connected to what the desired output of the model is. First, the regression problem where the goal is to find a relationship between a set of independent variables, and the dependent variables to identify what happens with the continous dependent output when an independent input variable changes or a new set of input data is provided. This is widely used in the field of prediction, forecasting and function approximation. A simple regression problem example is the pricing of a house given information of relevant specifications such as size, location and number of rooms. If the problem is to generate binary outputs, $y \in [0, 1]$, i.e. a yes or no analysis dependent on a input set, or to categorize a new observation to specific classes then the problem is mentioned as a classification problem. The same problem of determining the price of a house stated as a classification problem could be formulated by letting the algorithm determine if its pricey or cheap, by letting 0 correspond to pricey and 1 cheap respectively. Both regression and classification problems tend to use the learning

methodology supervised learning. If there is of interest to classify observations without determining its output classes then the corresponding problem with an unsupervised learning methodology is clustering. Other problems where unsupervised learning often is applied is to determine the unknown distribution of a data set or when a dimension reduction is of interest. (Haykin, 2009; Bishop, 2006)

Neural net specific learning task

Haykin (2009) enhances four general problems where modelling with ANN is relevant: Pattern Association, Pattern Recognition, Function Approximation and Control. A distinction of pattern recognition and pattern association is appropriate to avoid misnomer. Both of them are specialized forms of mapping input to a specified output by adjusting the weights of the network, where the learning methodology supervised learning is utilized.

Pattern recognition is also referred to as pattern classification and formally defined as the process where a received pattern/signal is assigned to specific binary classes. Thus the pattern recognition problem in the terminology of machine learning simply is a classification problem (Bishop, 2006; Haykin, 2009; Hagan et al., 2014).

Pattern association implicate utilizing a brain-like associative memory which consists of two basic phases:

- Storage phase, which refers to the training of the network that teaches it to associate similar input to the same output.
- Recall phase, which involves the retrieval of a memorized pattern in response to the presentation of a noisy version of input to the network. They are problems where the nets, for which involved weights are determined in such way that the net can store a set, M, of pattern associations. Thus, an important question for a pattern association problem is: how many patterns can be stored until the net starts to forget? (Fausett, 1994)

Function approximation also referred to as regression in the broader field, are type of problems where a net should approximate an unknown function, $f(\cdot)$ such that the mapping function, $g(\cdot)$, describing the mapping is realized by the network according to following formula

$$||g(X) - f(x)|| < \epsilon \tag{3.4.1}$$

where ϵ is a minor prespecified value.

Controlling problem aim to track a specific reference signal, y_{ref}, with the output, y by studying and minimizing an error signal, described as $\epsilon = y_{ref} - y$ where y is the output from the system. This means utilizing feedback of the output as input in the next time step to minimize ϵ, in other words controlling problem means to invert the input-output mapping. (Haykin, 2009)

Hagan et al. (2014) highlight 4 relevant problems as well: Function approximation, Pattern recognition, Clustering and Prediction where the first two is defined in the same way as by Haykin (2009) and clustering is defined as by the general machine learning problem.

Prediction is similar to the function approximation problem and falls under the categories of system identification, time series analysis or dynamic modelling. The basic idea is to predict future data points listed in time order. For example a control engineer might want to predict a future value of the concentration of some chemical. A dynamic neural network is vital to use in prediction problems. (Hagan et al., 2014)

Kaastra and Boyd (1996) emphasize two commonly used terms to describe a network: the neurodynamics and architecture where the combination along with the class define its paradigm. It will be obvious that the paradigm will mostly depend on the specific problem to solve, further Kaastra and Boyd (1996) conclude that there is an infinite number of ways to design a neural network and defining the learning task is a crucial part in order to succeed (Haykin, 2009; Hagan et al., 2014).

3.4.2 Neurodynamics

The way one artificial neuron processes data and how its inputs are combined into calculations to carry on information to another neuron is referred to as the neurodynamics of the network. McCulloch (1943) constructed the first artificial neuron with inspiration from of first order logic sentences, the neuron had the property of processing two binary inputs and generate a single binary output from its activation function. The model had significant constraints but could be used to implement boolean logic functions. Neural networks can be described as series of functional transformations where the transformation and mathematical calculations done from an ANN are a complex composition of the calculations from each node. To maintain non-linearity properties but still being model-free, not determining the setup of this complex composition, a neural network utilizes a functional relationships where its parameters are adaptive to the data. To understand these series of transformation one must understand the neurodynamics. To simplify the description of the neurodynamics, i.e. how one node operates, an illustration of a single-layer and one node ANN, with its including calculations, is illustrated in Figure 3.4.1.

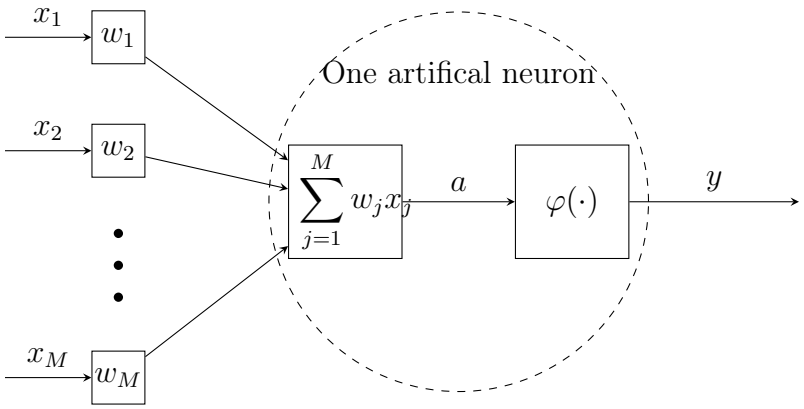

Figure 3.4.1: Example of an artifical neuron

First, a linear combination, called activation, $a(\mathbf{x}, \mathbf{w})$, of the input vector $\mathbf{x} \in \mathbb{R}^{M\times 1}$ and the synapses or weights from each input $\mathbf{w} \in \mathbb{R}^{M\times 1}$ are created as

$$a(\mathbf{x}, \mathbf{w}) = \sum_{j=1}^{M} w_j x_j. \tag{3.4.2}$$

Secondly, the activation are transformed to the output of the node, z, using a function, $\varphi(\cdot)$, called activation function by

$$y = \varphi(a(\mathbf{x}, \mathbf{w})). \tag{3.4.3}$$

Different activation functions can be utilized in different parts of the network. The activation function in the hidden layer is preferably non linear in such way that each output is a non linear combination of linear combinations of the inputs. A non linear property of the activation function is required to make advantage of a multilayer network, since the result of feeding a signal through two or more layer of linear processing activation functions are possibly obtained using a single layer, it follows from the fact that composing successive linear transformations is itself a linear transformation (Fausett, 1994; Bishop, 2006). Further, it is desirable that the activation function is differentiable to facilitate the training phase, which is described in Section 3.6. The activation function limits the amplitude to a preferred closed interval, typically between $[0, 1]$ or $[-1, 1]$ since it tend to increase stability while learning and also it is useful to maintain the normalization of the input data. Three of the most commonly used activation functions are the heavy side-, identity- and sigmoid function. The heavy side function or threshold function, φ_1, of some random input $x \in \mathbb{R}$ is described as

$$\varphi_1(x) = \begin{cases} 1 & \text{if} \quad x \geq 0 \\ 0 & \text{if} \quad x < 0 \end{cases} \tag{3.4.4}$$

and an illustration of the function can be seen in Figure 3.4.2.

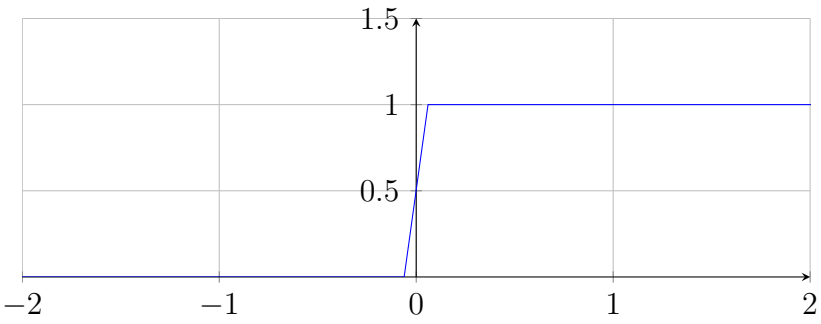

Figure 3.4.2: Heavyside step function

A neuron with this activation function is referred to as the McCulloch-Pitts model because of their pioneering work in the neurodynamic field. The properties of the model is that it generates a binary output $z \in [0, 1]$ (Haykin, 2009).

The identity function is a simple linear function through the origin, φ_2, of some random input $x \in \mathbb{R}$ and is described as

$$\varphi_2(x) = x, \quad \varphi_2(x) \in \mathbb{R}. \tag{3.4.5}$$

In order to limit the amplitude one could saturate the identity function to the ramp function, also known as the piecewise linear function, φ_2, of some random input $x \in \mathbb{R}$ which is described as

$$\varphi_3(x) = \begin{cases} 1 & \text{if} \quad x \geq 1 \\ x & \text{if} \quad 0 \leq x < 1 \\ 0 & \text{if} \quad x < 0 \end{cases} \tag{3.4.6}$$

and an illustration of the ramp function can be seen in Figure 3.4.3.

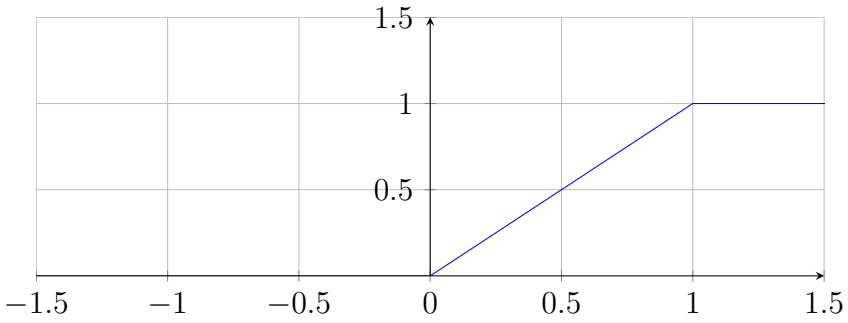

Figure 3.4.3: Ramp function

The function is sometimes referred to as the saturating linear function because the linearity is fixed to a specific interval (Zilouchian, 2001).

Sigmoid functions

The sigmoid functions is a family of functions having an S-shaped curve. The sigmoid functions is by far the most commonly used because of its characteristics of allowing non linearity and also being differentiable. Another advantage is that the derivative of the sigmoid function can be expressed in terms of the individual function itself which is useful when training the network (Zilouchian, 2001). Depending on preferred output, one could choose between two widely used sigmoid functions, the logistic sigmoid and the bipolar sigmoid functions. The logistic sigmoid function, φ_4, of some random input $x \in \mathbb{R}$, is preferable when the objective is to approximate functions that maps into probability spaces. The logistic sigmoid function is described as

$$\varphi_4(x) = \frac{1}{1 + e^{-\alpha x}}, \quad \varphi_4(x) \in [0, 1] \tag{3.4.7}$$

and is illustrated in Figure 3.4.4.

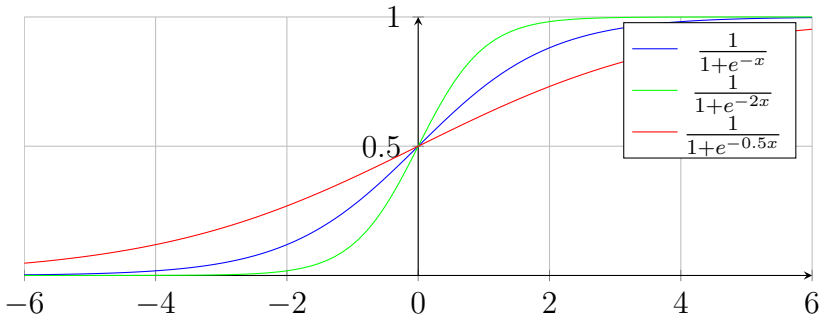

Figure 3.4.4: Logistic sigmoid function

where the parameter, α defines the slope of the function.

The hyperbolic tangent function (TanH) and hyperbolic tangent sigmoid function (TanSig) are rescaled versions of the logistic sigmoid to the range $[-1, 1]$, thus mentioned as the bipolar sigmoids. The advantage of these rescaled versions is that they suits data input around zero because the magnitude of the derivatives is greater for these values which enables a faster training. Further, using an activation function that outputs in the range $[0, 1]$ makes big negative values of the input saturating to zero which has a negative effect on the training because they get stuck in the current state and consequently the computational time increases. Another problem of only having positive values of the output is that all of the weights that feed into a node can only decrease or increase all together, in the training phase, for a given input pattern which creates an inefficient "zigg-zagging" update path (LeCun et al., 1998). TanH and TanSig are very similar where the steepness of the slopes differentiates them. As such, there is a trade off between getting a low computational time and an accurate mapping. However, when the computational time is not a major issue the TanH, φ_4, is mostly used and defined as

$$\varphi_5(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{3.4.8}$$

and is illustrated in Figure 3.4.5.

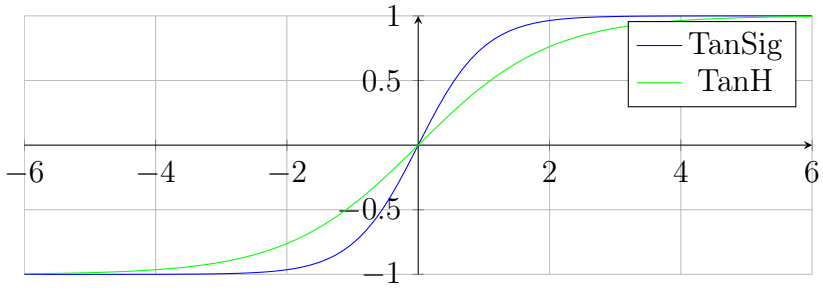

Figure 3.4.5: Bipolar sigmoids

Within the field of machine learning it is common to introduce a dummy variable, which most commonly is used modelling an neural network model as well. This is an

external node, x_0, with fixed value at $+1$ that along with its corresponding weight, w_0 allows for any offset in the transfer function and thus enables a shift of the function left or right depending on positive or negative value of w_0. Simplified, the purpose is to ensure that the activation function can produce all outputs no matter what the weighted sum of the input. E.g. if the sum of the weighted inputs is three, the bias node enables the neuron to still output 0. Figure 3.4.6 describes the shift of a logistic sigmoid function for different w_0.

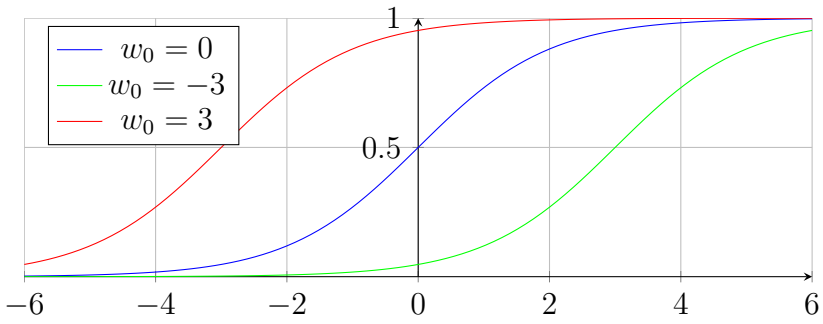

Figure 3.4.6: Sigmoid function

By expanding the input set with the bias node the activation introduced by (3.4.2) is reformulated to

$$a(\mathbf{x}, \mathbf{w}) = \sum_{j=1}^{M} w_j x_j + b w_0. \tag{3.4.9}$$

By defining the bias node as a new input, x_0 in the external input vector (3.4.9) is equivalent to

$$\begin{aligned} \mathbf{x} &:= \begin{bmatrix} x_0 \\ \mathbf{x} \end{bmatrix} \\ a(\mathbf{x}, \mathbf{w}) &= \sum_{j=0}^{M} w_j x_j. \end{aligned} \tag{3.4.10}$$

The expanded single layer network with a bias node is illustrated in Figure 3.4.7.

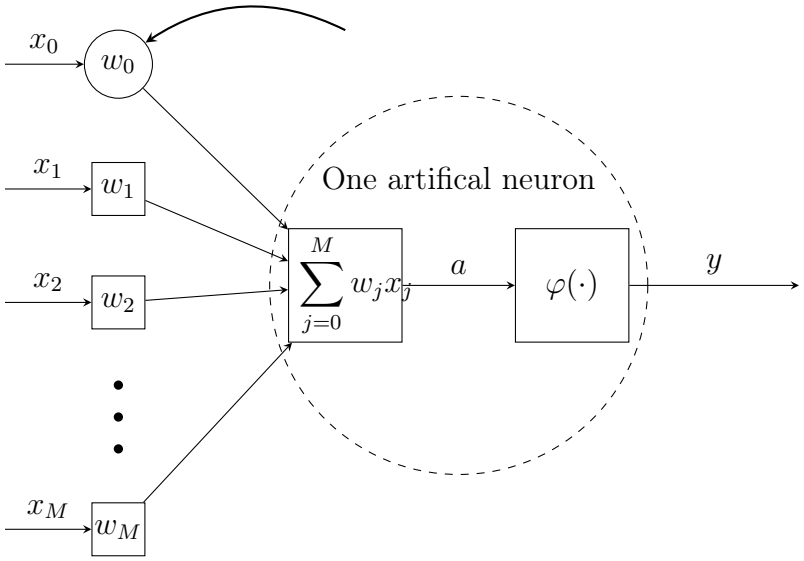

Figure 3.4.7: Artificial neuron with bias node

When expanding the single layer network, consisting of one individual node, to a multilayer network of L layers with M_l nodes in each layer the general artificial node is denoted as $h_j^{(l)}$. Further, the output of an artificial node, y, corresponds to an input, x_j, for the nodes in the next layer. Hence, we declare y_j as one of the final outputs of a network and give a more suitable representation of the output from node $h_j^{(l)}$ to $z_j^{(l)}$. The mathematical description of a random output would be a recursive formula of its activation function from previous outputs as

$$z_{j_l}^{(l)} = \varphi\left(\sum_{j_{l-1}=0}^{M_{l-1}} w_{j_{l-1},j_l}^{(l-1)} \varphi\left(\cdots \sum_{j_2=0}^{M_2} w_{j_2,j_3}^{(2)} \varphi\left(\sum_{j_1=0}^{M_1} w_{j_1,j_2}^{(1)} x_{j_1}(t) \right) \right) \right). \tag{3.4.11}$$

A graphical description of the general node within the general model of network would thus be as illustrated in Figure 3.4.8.

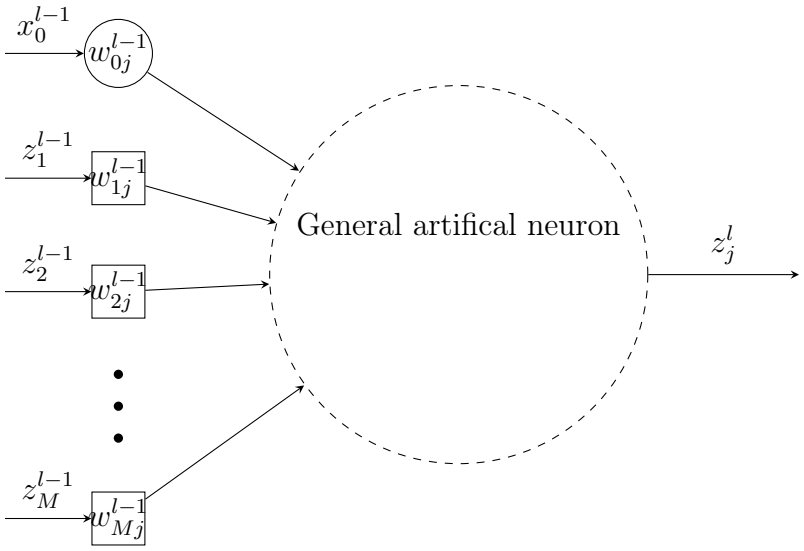

Figure 3.4.8: The general artificial neuron

Note that a bias node is added to each layer of the network.

3.4.3 Architectures

Putting multiple neurones together and a neural network is constructed, this arrangement of neurons into layers and how it is intimately linked is often referred to as the architecture of the network (Fausett, 1994). The total number of layers, L, can be categorized to three different type of layers, the input layer, hidden layer(s) and the output layer. Within each layer, l, the designer must decide how many nodes, M_l, each layer should consist of. Setting up the architecture when modelling the neural network thus requires determining lot of parameters.

Input

According to Kaastra and Boyd (1996) the number of nodes in the input layer, which in this report is denoted as a vector, $\mathbf{x} \in \mathbb{R}^{M_1 \times 1}$, is in the general case the most simple parameter to determine since each input domain after preprocessing, described later in this report, correspond to one neuron. For the casual problem, the number of input nodes is well defined and it is the number of independent variables associated with the problem. For instance if you should design a neural network for kite-flying conditions the inputs might be air temperature, wind velocity and humidity, i.e. three inputs (Hagan et al., 2014).

Determining what inputs to use when constructing neural networks describing financial time series is a harder nut to crack since the external specifications and the dependencies of the output is unknown. Researchers have utilized both technical data as well as fundamental data as inputs, where technical data is referred to as the processed data of the specific time series, e.g. time lagged returns, moving average, relative strength index and momentum etc., and fundamental data is the information from macroeconomics, microeconomics and business specific economy. i.e. posts from in-

come statements and the balance sheet. According to Zhang et al. (1998) one can not just choose a set of input variables arbitrarily and require the network to learn and predict a specific variable. The input set is the most critical decision variable to choose and the old adage "garbage in, garbage out" complies. This demands the designer to spend extensive time selecting input (Walczak, 2001). From the literature there is no consistent results indicating that some specific model is better than another. Some authors claim that the neural network is able to learn which inputs that matter and the weights are adjusted to zero if the input variable in question is redundant. Hence, they argue that the number of input is not crucial as long the inputs chosen is enough to provide sufficient information from all involving domains. Others advocate that too many input variables reduces the generalization performance because of the tendency of over-fitting the data. Also, the more inputs that is added the more computational complexity is added. Conversely too few inputs will under-fit the data, however both situations results in a bad out of sample generalization (Walczak, 2001; Zhang et al., 1998).

Knowledge-based Selection is a possible approach to determine what inputs that should be used. As described by Walczak (2001) it is a heuristic based on the belief that if the sufficient amount of information that explain the output is not given to the ANN, then the ANN cannot develop a correct model of the domain. The first step is thus to perform a standard knowledge acquisition which typically involves consultation with multiple domain experts, to guarantee that the set of inputs provide all relevant domains to the ANN. Second, the set is pruned to eliminate noise to the ANN in order to remove a possible risk of reducing generalization performance. This filtering is done with a correlation and dependence investigation. The method suggests to study the Pearson correlation matrix or alternatively making a chi-square test. The cutoff value for elimination must be determined in advance, but the Walczak (2001) suggests any correlation absolute value below 0.20 is probably a noise source to the ANN, and is hence to be removed. (Walczak, 2001)

Obviously there is a trade off between choosing less or more inputs. Hence some general properties of the inputs has been declared as desirable to identify the right number. Huang et al. (2004) describe that the inputs ideally should be uncorrelated with each other, but correlated with the output. This is based on the fact that inputs that are correlated with each other contain overlapping information that, casually speaking, will confuse the network. Dimension reduction is a method to achieve these properties from a large set of inputs by decreasing the number of inputs and limiting the reduction of the information provided to the network. Following this, two methods for dimension reduction is described.

Autocorrelation Criterion is a method developed by Huang et al. (2004) based on the investigation of correlation between inputs itself and to outputs. The model creates different input information from the same data series, by using different lag periods. It is useful to obtain a partial description of the time series since the coefficient describes the degree of correlation between different data observations. The autocorrelation

coefficient can be described as

$$\varrho_k = \frac{\sum_{t=k+1}^{T}(y_t - \bar{y})(y_{t-k} - \bar{y}_{t-k})/(T-k)}{\sum_{t=1}^{T}(y_t - \bar{y})^2/T} \tag{3.4.12}$$

where $\bar{y}$ is the sample mean of y i.e.

$$\bar{y}_{t-k} = \frac{1}{T-k}\sum_{t=k+1}^{T} y_t. \tag{3.4.13}$$

The model is based on the theory that the contribution of each input is dependent on the other inputs. Each of them should be predictive but should not be correlated to the other input variables since they provide the same information and thus would degrade the ANN performance. The network will be confused by not knowing of which input to use and may alternate back and forth. Hence, the method is based on following two assumptions:

1. The inputs have as high degree of correlation to the output as possible
2. The inputs have as low degree of correlation to each other as possible

Therefore the algorithm to determine which lag periods, $a(1), a(2), \cdots, a(m)$, to use is based on the following three steps:

- Step 1. Set upper limit of lag period N. Let $a(1) = 1$ and $m = 1$
- Step 2. $a(m+1) = \arg\max_k \frac{|r_k|}{\sum_{i=1}^{m}|r_{k-a(i)}|}$
- Step 3. Let $m = m + 1$. If $a(m)$ is less than N, go to Step 2. Otherwise, exit

This approach is useful for many practical problems, as it is easier to have data then to determine theoretical guesses about the underlying laws governing the structure of the data. Further, it avoids long experimentation in the input selection phase. (Huang et al., 2004)

Principal Component Analysis (PCA) is a preprocessing method that can be used to obtain k linearly uncorrelated variables stemming from n possibly linearly correlated variables. The idea is to reduce the number of variables in a model, but still keep as much information as possible from the original variables, while at the same time having orthogonal variables. (Hull, 2012) By definition, the covariance matrix $\mathbf{C}$ to some input $\mathbf{x}$ is symmetrical, and can hence be rewritten as $\mathbf{C} = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^{-1} = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^T = \sum_{i=1}^{n}\lambda_i \mathbf{q_i}\mathbf{q_i^T}$ where $\mathbf{Q}$ is a matrix consisting of eigenvectors $\mathbf{q}_i$ of $\mathbf{C}$ and $\mathbf{\Lambda}$ is a diagonal matrix with the eigenvalues λ_i of $\mathbf{C}$. (Blomvall, 2016) The eigenvalues are computed by solving the characteristic polynomial of $\mathbf{C}$ which is given by $det(\mathbf{C} - \boldsymbol{\lambda}\mathbf{I}) = 0$, where $det(\cdot)$ is the determinant and $\mathbf{I}$ is the identity matrix. The corresponding eigenvector $\mathbf{q}_i$ is computed by solving the characteristic polynomial using the corresponding eigenvalue λ_i. (Janfalk, 2014) In PCA the eigenvectors and eigenvalues are sorted in descending order, i.e. $\lambda_i \geq \lambda_{i+1}$, and as such the first eigenvector has the highest degree of explanation of the eigenvectors in $\mathbf{Q}$.

The covariance matrix can now be estimated with the first k eigenvectors $\mathbf{Q}_k$ with corresponding eigenvalues $\mathbf{\Lambda}_k$

$$\mathbf{Q}_k = \begin{bmatrix} | & & | \\ \mathbf{q}_1 & \dots & \mathbf{q}_k \\ | & & | \end{bmatrix} \quad \mathbf{\Lambda}_k = \begin{bmatrix} \lambda_1 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \lambda_k \end{bmatrix} \tag{3.4.14}$$

by $\mathbf{C} \approx \mathbf{Q}_k \mathbf{\Lambda}_k \mathbf{Q}_k^T$. Based on the orthogonality of the eigenvectors this will explain a fraction of the variance according to

$$\frac{\sum_{i=1}^{k} \lambda_i}{\sum_{i=1}^{n} \lambda_i}. \tag{3.4.15}$$

The principal components $\boldsymbol{\kappa}_{it}$ for eigenvector i for input $\mathbf{x}_t$ at time step t can now be computed as

$$\boldsymbol{\kappa}_{it} = \mathbf{q}_i^T \mathbf{x}_t, \quad Cov(\boldsymbol{\kappa}_i, \boldsymbol{\kappa}_j) = \begin{cases} 0 & \text{when } i \neq j \\ \lambda_i & \text{when } i = j \end{cases}. \tag{3.4.16}$$

(Blomvall, 2016)

Hidden layers

The layers, $l \in (2, 3, \dots, L-1)$, between the input and output layers of a multilayer network is called hidden layer(s). Those consists of the hidden nodes where each layer becomes a vector of hidden nodes. The term "hidden" refers to the fact that the layer is not seen from either the input or the output layer. By adding the number of hidden layers the network is enabled to extract higher order statistics from its polynomial which provides the network with its ability to generalize. But this does not imply that simply adding an extra layer will improve the network generalization ability. It is a vital decision setting the size and scope of the network, a bigger network results in a more complex training. Another consequence of too many hidden layers is the risk of getting an overfitted model. (Kaastra and Boyd, 1996; Haykin, 2009) Thus, throughout the history, the researchers have been stating the question: "*What is the minimum number of hidden layers in a multilayer network that provides an approximate realization of any continuous mapping?*"

The answer was stated by Hornik et al. (1989) who formulated the Universal Approximation Theorem, (UAT), which basically says that any continuous function can be approximated through a feedforward network with only one single hidden layer.

Theorem 3.4.1 (Universal Approximation Theorem) *Let $\varphi(\cdot)$ be a non constant, bounded, and monotone- increasing, continuous function. Let I_{m_0} denote the m_0-dimensional unit hypercube $[0,1]^{m_0}$. The space of continuous functions on I_{m_0} is denoted by $C(I_{m_0})$. Then, given any function $f \in C(I_{m_0})$ and $\epsilon > 0$, there exists an integer m_1 and sets of real constants a_i, b_i and $w_{i,j}$ where $i = 1, 2, \dots, m_1$ and $j = 1, 2, \dots, m_0$ such that we may define*

$$F(x_1, x_2, \cdots, x_{m_0}) = \sum_{i=1}^{m_1} a_i \varphi\left(\sum_{j=1}^{m_0} w_{ij} x_j + b_i \right)$$

as an approximate realization of the function $f(\cdot)$*; i.e.,*

$$|F(x_1, x_2, \cdots, x_{m_0}) - f(x_1, x_2, \cdots, x_{m_0})| < \epsilon$$

$\forall \quad x_1, x_2, \ldots, x_{m_0}$ *that lie in the input space.*

Along with Cybenko (1989) he proved his theorem a few years later with a focus on the dependency of the activation function and concluded that there are very general conditions on the activation function to enable the theorem. Moreover it should be non-constant, bounded, continuous and monotonic increasing Hornik (1991).

This behaviour of approximating a continuous function with a set of other continuous functions is similar to the Fourier series approximation. Anyhow, the theorem proves one layer is enough but does not express that it necessarily is the optimal choice in sense of generalization capability (Haykin, 2009).

Since the UAT has been proven one could question why anyone would want to add an extra layer and increase the risk of overfitting the data? Anyhow there are researchers that argue for multiple number of hidden layers. Haykin (2009) state that there is a problem with just one hidden layer since the neurons tend to interact with each other, consequently he concluded the following statements

1. Local features are extracted in the first hidden layer. Specifically, some neurons in the first hidden layer are used to partition the input space into regions, and other neurons in that layer learn the local features characterizing those regions.
2. Global features are extracted in the second hidden layer. Specifically, a neuron in the second hidden layer combines the outputs of neurons in the first hidden layer operating on a particular region of the input space and thereby learns the global features for that region and outputs zero elsewhere.

Chester (1990) advocate that for some problems a small, two hidden layer network can be used since a single hidden layer network will require large number of nodes. He claim that there is a trade off between adding extra nodes in the first hidden layer and expanding to more hidden layers. He argues that doing the right trade off is important when approximating functions with lot of "hills or valleys". Each additional unit in a potential second layer enables the network to learn with a relative few number of units in the first layer.

Unfortunately there is a limited discussion in the literature regarding methods to determine the optimal number of hidden layers and the size of them, previous researches often utilize "trial and error", which consequently shaped the standard procedure: start with a network with one hidden layer. If the performance of the is not satisfactory, then an extra hidden layer is added. But one should be careful when adding more hidden layers, it is unusual since the training becomes more difficult. The reason is that each layer perform a squashing operation because of the activation function which causes the derivatives of the performance function with respect to their weights

in the early layers to be quite small that consequently slows down the convergence of steepest descent optimization. (Hagan et al., 2014)

Hidden neurons

As well as with the number of hidden layers, finding the optimal number of hidden neurons is a prioritized area of research. One fact though, is that a function with a large number of inflection points requires a large number of neurons in the hidden layer(s) (Hagan et al., 2014). According to Zhang et al. (1998) it is the hidden nodes in the hidden layer(s) that allows neural networks to detect the feature, capture the pattern, and to perform complicated nonlinear mapping between input and output. For instance, the number of hidden units in a layered feedforward network can play a similar role to the number of terms in a polynomial (Bishop, 1995).

The trade off between choosing a larger network versus a smaller remains being a unsolved issue in the literature. More hidden units results in a longer learning time of the characteristics whilst smaller network may become trapped into a local error minimum due to lack of generalization possibilities. The lack of generalization increases required time to train but on the other hand there is an almost linear correlation between the number of hidden units and the number of samples required for the training process. Larger networks require more training examples than small networks to achieve the same generalisation performance. Further, too large network open up the risk for overfitting (Haykin, 2009; Bishop, 2006; Hagan et al., 2014; Kavzoglu, 1999).

Lots of rules of thumb have been proposed and most commonly discussed is that there is a dependency of the size of input nodes, M_1. Different proposals, from different authors, claim that the number of hidden nodes used should be $2M_1 + 1$, $2M_1$, M_1 or $\frac{M_1}{2}$ (Zhang et al., 1998). But one could argue that this is nonsense since the results approximating different functions is inconsistent and according to Sarle (2002) these methods are not involving any thoughts of number of training examples, amount of noise in the target and the complexity of the function. Consequently, this leaves the designer to try different heuristics based on trial and error. Two general approaches are discussed: constructive techniques and pruning. The first suggests to start with a smaller network and iteratively increase the number of hidden nodes in the layer(s) until satisfactory learning is achieved. This might result in problems since the there are problem that smaller network are more sensitive to weight initialization and other parameters and thus the network might be trapped in local optimum. The other approach, pruning, is an opposite method where one starts with a larger network and iteratively decreases the number of nodes after the training by studying the interconnections to find out which nodes are redundant (Kavzoglu, 1999).

Even though a trial and error method is used there is of importance to decrease the number of architectures tried to reduce training scope, hence setting limits of hidden units tried is of interest. Bishop (2006) conclude that having a number of hidden units smaller than the number of input units in the first hidden layer makes the transformations that the network can generate not the most general possible because information from the input is lost in the dimensionality reduction.

Unfortunately, UAT is not constructive with the number of hidden units as with the hidden layer(s), it only says that the theorem holds for a unlimited size of one hidden layer (Haykin, 2009). Determining an upper limit though, one could keep the Russian mathematician Kolmogorov's theorem from 1957 in mind. In neural network terms this theorem says that any continuous function $y(x)$ from a number, n, inputs of x_i to an output variable y can be represented exactly by a three-layer neural network having by having, $2n$, units in the first hidden layer and, n, units in the second hidden layer. The theorem has been slightly modified throughout the years. The theorem has been constructive proven and is formulated as

Theorem 3.4.2 (Kolmogorov's theorem) *Let $f : \mathbb{1}^n := [0, 1] \mapsto \mathbb{R}$ be an arbitrary multivariate continuous function. Then it has the representation*

$$f(x_1, x_2, \ldots, x_n) = \sum_{q=0}^{2n} \Phi_q \left(\sum_{p=1}^{n} \psi_{q,p}(x_p) \right)$$

with continuous one-dimensional outer and inner functions Φ_q and $\psi_{q,p}$. All these functions Φ_q, $\psi_{q,p}$ are defined on the real line. The inner function $\psi_{q,p}$ are independent of the function f.

(Braun and Griebel, 2009)

Output

The number of output nodes is relatively easy to determine since it is often stated with the task and as with the input nodes set by the external specifications (Hagan et al., 2014). Unravelling whether the problem encountered is a classification or regression problem makes the choice of number of output nodes easier. The number of output nodes in a classification problem is a direct consequence of the number of classes or labels the selected input should be divided into. By using more than one output node and create combinations of the binary values from each output one could extend the classification network to classify into more than two classes. (Bishop, 2006; Ng, 2016) There is no general method of determining the number of outputs of a regression problem, it is given by the specific problem. Within prediction the time horizon has a high influence since there basically are two ways of forecasting, one step ahead and multiple step ahead. The latter way could be done in two ways, either by iteratively letting the first step prediction become an extra input to the second step or by utilizing multiple output nodes one for each time horizon. (Zhang et al., 1998) Another common context where a multiple output might be useful is where the problem is complicated and involves determining values of different domains, but these networks are according to Kaastra and Boyd (1996) strictly avoided. The authors claim that widely spaced outputs will cause inferior problems. Neural network training is based on minimizing the average error of all outputs, for example the network could generate forecasts 1- and 6-month ahead, the problem that arises is that the network will minimize the biggest prediction error which most certainly will be the 6-month prediction. As a result, a relatively large improvement will not be done on the 1-month prediction if it increases the absolute error with a value that is greater. An undesirable property attempting to predict with the best accuracy as possible.

Obviously the size of the architecture is crucial but as in all modelling problems, when designing the architecture one should always have in mind to not use a bigger network when a smaller network will work. Also, the number of training examples limits the size that should be used, as a bigger network add to the number of free parameters (weights). Zhang et al. (1998) provide a rule of thumb that each weight in the network should have at least 10 training examples, which consequently means that the more inputs, layers and nodes that are used, the more training data should be available. Hence, many researchers advocate to follow the principle attributed to the English logician William of Ockham referred to as Ockham's Razor. The principle is formulated as

"Accept the simplest explanation that fits the data"

The fundamental idea is that increased complexity increases the possibility for errors. (Haykin, 2009; Hagan et al., 2014)

3.4.4 Classes of network

Another vital part defining the paradigm of the neural network is determining its class. There are basically no limitations in how one could design the neural network class but Haykin (2009) emphasizes two fundamental classes that are most commonly used by researchers: the feedforward and the recurrent network. These classes can bee seen as basis that in some way are tweaked in the other less frequently used classes.

Dynamic and static networks

Before introducing the classes of networks we introduce two class categories. ANNs can be classified into static or dynamic networks, where the static ones are calculated directly from its input. In a dynamic network on the other hand the output depends not only on the current inputs, $\mathbf{x}(t)$ but also on the previous input, $\mathbf{x}(t-1)$. Hence we introduce the unit delay operator in Figure 3.4.9

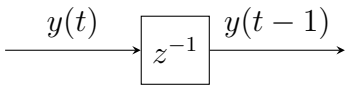

Figure 3.4.9: Delay block

The output of the block is the input delayed by one time step, assuming that time is updated in discrete steps and takes only integers value. Using multiple delay operators in one block enables the designer to build a dynamic network that utilizes input from different time steps. Using a tapped delay line at the input basically means extending the input vector with delayed values of the present input. Hence, a figure describing the tapped delay line is introduced in Figure 3.4.10. (Hagan et al., 2014)

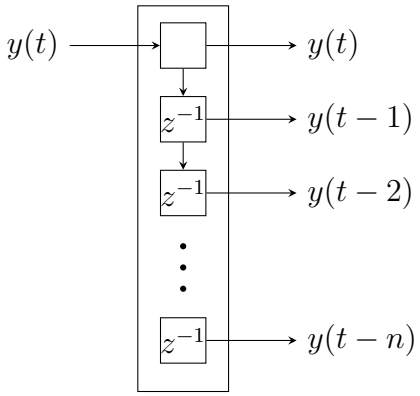

Figure 3.4.10: Tapped delay line

A dynamic network is said to have a "memory". Its response at any given time will depend not only on the current input, also the history of the input sequence are taken into account. Hence a dynamic network has applications in areas as control of dynamic systems, prediction of protein structures in genetics and prediction in financial markets (Hagan et al., 2014)

Feedforward

The first class to be introduced is the feedforward neural network (FFNN) that got its name from the direction of the information flow i.e. the connections of the node does not form a cycle. FFNN is the most successful class and the class that have been most studied throughout the years, it is also the class refered to when the important UAT and Kolmogorov's theorem were defined. As categorization of FFNN the literature often separates single-layer feedforward neural networks (SLFFNN) and multi-layer feedforward neural networks (MLFFNN) where the name tells the difference, a multi-layer feedforward network consists of at least one hidden layer.

Single-layer feedforward is the oldest and simplest neural network and the most known are the McCulloch-Pitts-neuron, Perceptron and Aldaline network, which all have in common that they consist of one single neuron. McCulloch (1943) developed the first, which basically produced a logic output, $y \in [0, 1]$, dependent on if the sum of all weight corrected inputs become higher than a specific threshold, θ. The latter two are basically smaller extensions where the Perceptron enables the weights to turn negative, and create a bipolar output, $y \in [-1, 1]; \neq 0$ dependent on if the sum of all weight corrected inputs become higher than θ, but the foremost development is that the Perceptron enables a learning rule. Further, the Aldaline network was introduced by the professor Bernard Widrow and his student Ted Hoff 1960 where they expanded the features of the Perceptron by enabling a linear transfer function instead of a heavy side function and also introduced the least mean square algorithm as learning rule (Hagan et al., 2014). However, all these single layer networks lack functionality in more complex problems.

Multi-Layer feedforward The key characteristic of the single-layer perceptron is that it creates linear relationships. What if we have a non-linear pattern or categories

that cannot be separated by linear boundaries? To deal with such characteristics, the multi-layer feedforward network is introduced, and illustrated in Figure 3.4.11. (Hagan et al., 2014)

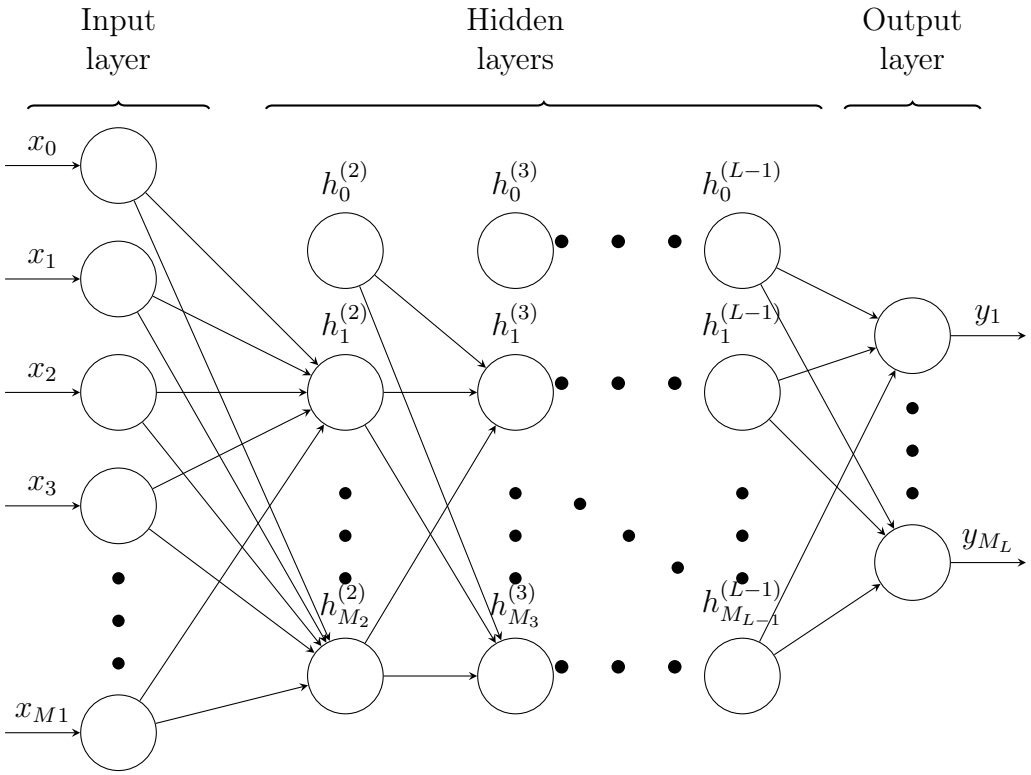

Figure 3.4.11: Example of a common neural network architecture, namely a feedfoward network

The most successful model in the context of pattern recognition, function approximation and prediction is known as the Multi-Layer Perceptron, (MLP). In fact, "Multi-Layer Perceptron" is a misnomer, because the model comprises of multiple layers of models with continuous non-linearities rather than multiple Perceptrons with discontinuous non-linearities. This means that the neural network function is differentiable with respect to the network parameters, and this property will play a central role in network training. The terminology in the literature is inconsistent and the authors often refer to a general multilayer network when talking about a MLP or could also name it as e.g. Kaastra and Boyd (1996) does, namely as the backpropagation (BP) architecture. The name origins from its successful training methodology using gradient descent applied to a sum-of-squares error function. A more comprehensive description of BP will be in Section 3.6. (Bishop, 2006) The multilayer perceptron trained by BP is currently most widely used (Bishop, 2006; Haykin, 2009; Hagan et al., 2014).

The standard neural network architecture for function approximation problems is the MLP, with Tansig activation functions in the hidden layers, and linear activation functions in the output layer. Further, a tweak of an MLP is useful in prediction problem, mentioned as the focus time-delay neural network, i.e. an MLP with tapped

delay line at the input. (Hagan et al., 2014)

Recurrent network

Contrary to the FFNN the Recurrent Neural Network (RNN) is a model with bi-directional data flow. An RNN is a network with feedback, i.e. the outputs of the nodes of the network are sent back to be used as inputs. This feedback can be done in different ways: by using self-feedback, i.e. the final outputs are used, or through usage of some or all of the hidden nodes, for which then the network is mentioned as a fully connected recurrent neural network (FCRNN), or also by using both FCRNN and self-feedback, which is illustrated in the Figure 3.4.12.

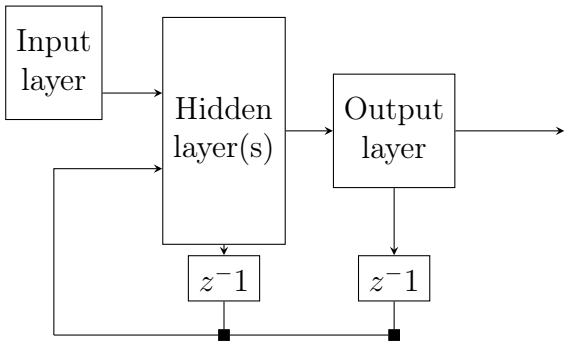

Figure 3.4.12: Example of a recurrent neural network with feedback from both hidden and output layer(s)

One of the first FCRNN is the famous Hopfield network, invented by Hopfield (1982) which later also were extended to the "noisy Hopfield network" by David H. Ackley and Sejnowski (1985) and officially named as the Boltzmann machine. Both of them are networks with associative memories that guarantees the network to find a local minima and have been successful within physics. The prior one is though limited by a storing capacity of association and the second suffer from having a painfully slow training phase. (Haykin, 2009; Hagan et al., 2014) However, the computational capability of FCRNN is summarized into a theorem stated by Siegelmann and Sontag (1991).

Theorem 3.4.3 *All Turing machines may be simulated by fully connected recurrent networks built on neurons with sigmoidal activation functions.*

A Turing machine, invented by Alan Turing 1936, with a basic description, is a machine that is capable of computing any computable function, it could be seen as the opposite to the neural networks in the domain of learning since it has infinite programs stored. (Haykin, 2009; Fausett, 1994) General RNNs are used as associative memories, in which stored data is recalled by association with input data, rather than by an specific address (Hagan et al., 2014).

Nonlinear Autoregressive model with exogenous input (NARX) is one of the most commonly mentioned RNNs. It is a dynamic recurrent network, that enclose several static layers of the network. The NARX model is based on the linear ARX model,

which is commonly used in time series modeling, but the NARX enables nonlinearity. (Hagan et al., 2014) The NARX-model is primarily characterized by using

- Present and past values of the input vector, the exogenous input, $\mathbf{x}(t)$, i.e. $\big(\mathbf{x}(t), \mathbf{x}(t-1), \cdots, \mathbf{x}(t-\tau_{lag})\big)$ by utilizing a tapped delay line.
- Delayed values of the output vector, autoregression, $\mathbf{y}(t)$, i.e. $\big(\mathbf{y}(t), \mathbf{y}(t-1), \cdots, \mathbf{y}(t-\tau_{fb})\big)$ by utilizing feedback.

Hence, the behavior of the NARX network is described by the function

$$\mathbf{y}\big(t\big) = f\big(\mathbf{y}(t-1), \mathbf{y}(t-2), \cdots, \mathbf{y}(t-\tau_{fb}); \mathbf{x}(t), \mathbf{x}(t-1), \cdots, \mathbf{x}(t-\tau_{lag})\big) \quad (3.4.17)$$

where, f, is a nonlinear function of its arguments (Haykin, 2009). Siegelmann et al. (1997) provided a second theorem which reflect the computational power of a NARX network with respect to other FCRNN.

Theorem 3.4.4 *NARX networks with one layer of hidden neurons with bounded, one-sided saturated activation functions and a linear output neuron can simulate fully connected recurrent networks with bounded, one-sided saturated activation functions, except for a linear slowdown.*

The opinion of the recurrent networks are inconsistent between the researchers, and its obvious that it suits some problems better than others. According to Hagan et al. (2014) a NARX network is well suited to function approximation and prediction problem whereas Kaastra and Boyd (1996) claim that a recurrent network is less common in time series forecasting.

Hagan et al. (2014) express that general RNNs are potentially more powerful than feedforward networks, since they are able to recognize and recall temporal, as well as spatial, patterns. If a network does not have any feedback connections, then only a finite amount of history will affect the response. But there are downsides too. A recurrent network involves more complexity because, in contrast to FFNN, the output is a function not only of the input but also a function of time which involves a functional instability. This means that the response of the network may converge to a stable output, but could as well oscillate, follow a chaotic pattern or explode to infinity. Further, the training part of a recurrent network is said to be more difficult. One could visualize an RNN as an FFNN unfolded in time, if 5 time steps of feedback is used this could be seen as FFNN with 5 layers, one for each step. Suppose a sigmoid activation function is used, then if the output is near the saturation point of any time point, the resulting gradient will be quite small, which impair the training algorithm. (Hagan et al., 2014)

Ensemble averaging

According to Haykin (1999) the method of ensemble averaging is a type of static structure committee machine. A committee machine is a model that combine a number of neural network outputs, to derive the final output. The idea is that the combination of several neural networks supposedly derive at a superior solution to a solution derived

by any of the individual networks. The category of ensemble averaging linearly combines the outputs of the individual neural networks, in contrast to dynamic structures (see Haykin (1999)) that combines the individual networks non-linearly. Figure 3.4.13 illustrates a general ensemble averaging model.

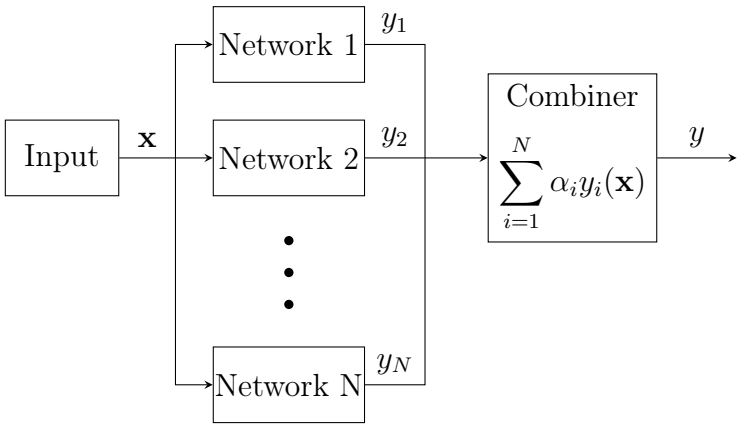

Figure 3.4.13: A general model for ensemble averaging, where $\mathbf{x}$ is the input, y_i is the output from network i, and $\alpha_i \geq 0$ is a weight for the output from network i

Consider a sample of neural network as regression estimators y_i that estimate the desired output with a mean squared error on the validation set $MSE[y_i]$. Perrone (1993) describe that it is unsatisfactory to choose the commonly naive estimator of the problem, i.e. to choose the neural network that estimates the output with y_i that satisfies

$$y_{NAIVE}(x) = \arg\min_{y_i} MSE[y_i]. \tag{3.4.18}$$

This is based on two major reasons. The first reason is that the rejection of estimators y_i potentially discards useful information stored in the discarded estimators. The second reason is that since the validation set is random, there is a certain probability that another neural network than the naive estimator will perform better on a different sample of the validation set based on the same distribution. As such, Perrone (1993) define basic ensemble model (BEM) of N estimators as

$$y_{BEM}(x) = \frac{1}{N} \sum_{i=1}^{K} y_i(x) \tag{3.4.19}$$

which under the assumption of mutually independent regression errors will reduce the mean squared error to $\overline{MSE} = \frac{1}{K} \sum_{i=1}^{K} MSE[y_i]$. This means that the ensemble average will keep the bias (training error) but reduce the variance (generalization error) of the individual networks. Haykin (1999) conclude that in ensemble averaging, the networks should be purposely overtrained since the averaging will reduce the variance of the final network, but keep the bias of the individual networks. As such, combining ensemble averaging and regularization and/or early stopping is by Haykin (1999) adviced against. Also, Naftaly et al. (1997) advice against varying all the hyper-parameters

(architecture, training set etc.) of the ensemble averaging networks, and instead argue to only vary the weight initialization to reach best performance.

3.5 Data preprocessing

Preprocessing the input and output data of a neural network to end up with an adequate data representation is crucial in order to design a successful network. The aim is to remove noise, highlight relationships, flatten the distribution and detect trends to help the neural network detect relevant patterns. (Kaastra and Boyd, 1996) This coincide with the motivation of determining the right inputs, hence the choice of inputs and preprocessing technique walks hand in hand. For example, the PCA technique is a preprocessing technique described in Section 3.4.3. While the PCA transforms the data (preprocessing), it will lower the dimensionality, i.e. the number of inputs and subsequently the eigienvectors will be the inputs. According to Han et al. (2012) there are several data preprocessing techniques, ranging from data cleaning (removal of noise and inconsistencies), data integration (merging data from different sources), data reduction (aggregating or eliminating redundant data) to data transformation (e.g. scaled into a certain range). The techniques are not mutually exclusive and may work well together in preprocessing purposes.

A common technique in data transformation is to normalize the data. Normalization of the data will remove the dependence of measurement unit and will transform different data types into a common range, often $[-1, 1]$ or $[0, 1]$. (Han et al., 2012) A general model for normalizing a data point x to x', where the minimum value of x is min_x and the maximum value of x is max_x, into a range $[newMin_x, newMax_x]$ is

$$x' = \frac{x - min_x}{max_x - min_x}(newMax_x - newMin_x) + newMin_x \tag{3.5.1}$$

which for the case of normalized data into $[0, 1]$ simplifies to

$$x' = \frac{x - min_x}{max_x - min_x}. \tag{3.5.2}$$

If the min-max normalization is dominated by outliers, i.e. the majority of data points are transformed to a small portion of the min-max interval a zero-mean normalization might come handy of computational efficiency reasons described in Section 3.4.2 and especially in Section 3.4.2 (Han et al., 2012). The zero-mean normalization is defined by

$$x' = \frac{x - \bar{x}}{\sigma_x} \tag{3.5.3}$$

where $\bar{x}$ is the arithmetic mean of the sample and σ_x is the sample standard deviation. To further reduce the effect of outliers, zero-mean normalization might be used with mean absolute deviation instead, such that

$$x' = \frac{x - \bar{x}}{s_x} \tag{3.5.4}$$

and

$$s_x = \frac{1}{n}(|x_1 - \bar{x}| + |x_2 - \bar{x}| + \ldots + |x_n - \bar{x}|). \tag{3.5.5}$$

(Han et al., 2012)

By preprocessing the data into a proper data representation, Kaastra and Boyd (1996) claim that the performance of neural networks may be enhanced as it can specialize in its proper objective. A useful tool in preprocessing is histograms, which can reveal if the variables are scaled appropriately (Kaastra and Boyd, 1996). In addition to improving the quality of the pattern recognition, the data preprocessing might reduce time required for computation as the gradients in the backpropagation algorithm becomes better suited for the problem (Han et al., 2012).

3.6 Optimization of neural network parameters

The optimization of the neural network parameters consists of two major parts. The first part is the optimization of the neural network weights given a specific set of hyper-parameters (or paradigm). This part is called neural network training. Training of a neural network is hence the process of calculating the set of weights, $\mathbf{W}^*$, that minimizes the error function of the network for a given set of training examples. This corresponds to the calculation of the coefficients in a linear regression model described in Section 3.2.2. Training examples are samples of inputs and outputs that the neural network will use to recognize patterns, and therefore generalize. The second part is the optimization of the hyper-parameters, i.e. determining which neural network paradigm that gives the best generalization ability.

This section will introduce methods for training the individual neural networks, but also methods for optimizing the hyper-parameters.

3.6.1 Purpose of network training

The training of a neural network refers to the optimization of the weights to minimize the error function of the network. However, one is usually interested in optimizing the generalization ability of the network (Prechelt, 1998), which might not coincide with the minimum of the error function due to different in sample and out of sample data. Most researchers' purpose when training the network is to achieve a high generalization ability (see e.g. Prechelt (1998); Bishop (2006); Haykin (2009)). Therefore, it may not be of interest to find the global optimum of the error function, which is further discussed in Section 3.6.7.

3.6.2 Training, validation and testing data sets

The available data set of the past observations of the relationship between input and output is usually partitioned into different subsets, namely training, validation and training sets. The mentioned training examples, that is used to optimize the portfolio

weights, is found in the training data set. The validation data set consists of data that different paradigms of networks can be validated on, so that the best configuration can be chosen. The testing set has the purpose of being a truly untouched out-of-sample data set so that the final model can be evaluated on data corresponding to future data. Usually the training set is the largest of the sets. The partitioning of the data can be done in different ways. One example is to chronologically divide the data set into the three subsets. For estimating expected value in finance, some researchers propose that the data should be randomly partitioned into the different subsets, with the purpose of validating models and testing the final model regardless of market trend. Kaastra and Boyd (1996) provide an example, where they suppose that the testing set consists of data where the market is bullish. A model that has low generalization error on this test set might be considered a good model, but might actually just be performing good on a market uptrend, and not during general market conditions, hence having a bad generalization ability.

3.6.3 Testing

Once the final model is decided upon, the generalization ability is tested on a previously defined but untouched testing set. Many researchers, e.g. Kaastra and Boyd (1996); Zhang et al. (1998), stress the importance of by no means using this untouched testing set while selecting model to have a final independent check and evaluation of the neural network generalization performance before finally using it. If the entire data set in drawn from the same population, the researcher might just commit a portion of the data to be a testing set. However, if the drawn data comes from a non-stationary times series the conclusions drawn from the testing might differ greatly with the actual generalization performance. Instead, to simulate a situation where the neural network is retrained occasionally, a walk-forward testing routine might be utilized instead. (Kaastra and Boyd, 1996; Hu et al., 1999)

Walk-forward testing routine

Kaastra and Boyd (1996) describe a walk-forward testing routine being popular in commodity trading systems see Figure 3.6.1. The data is divided into subsets of overlapping training, validation and testing sets. The size of the testing set will control the frequency that the network can be retrained at. In each of the training, validation and testing sets the network weights will be stationary, but looking at the entire times series, the network weights will be able to model a non-stationary functional relationship. However, by conducting this technique the assumption is made that the researcher will retrain the network at some frequency going forward, in order for the error on the testing period to be a fair estimate of the generalization ability of the model. The size of the testing set will determine how often the network can be retrained.

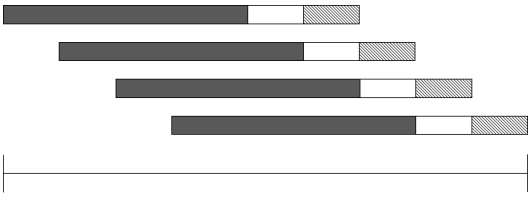

Figure 3.6.1: A moving window walk-foward testing routine

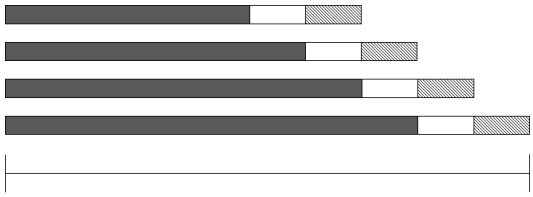

Figure 3.6.2: An expanding window walk-foward testing routine

Hu et al. (1999) describe an additional moving forward testing scheme, that rather than moving the window forward, expands it, see Figure 3.6.2.

3.6.4 Error functions

In order to evaluate the performance of the network in the training, validation and testing set respectively, an error function, $J(\cdot)$, has to be defined for the data sets. We will refer to training set error (self-explanatory), generalization error (error in the validation set) and test set error (self-explanatory). Three common metrics are the mean absolute error (MAE), the mean squared error (MSE) and the mean of the fourth power error (MFPE) defined as

$$J_{MAE} = \frac{1}{N}\sum_{m=1}^{N}|\hat{y}_m - y_m|, \quad J_{MSE} = \frac{1}{N}\sum_{m=1}^{N}(\hat{y}_m - y_m)^2, \quad J_{MFPE} = \frac{1}{N}\sum_{m=1}^{N}(\hat{y}_m - y_m)^4 \tag{3.6.1}$$

where $\hat{y}_m$ is the output, y_m is the desired output for training example $m = 1, 2, \ldots, N$ (Bishop, 2006; Haykin, 2009; Gangal et al., 2007). As the exponent gets higher, the more the bigger errors will influence the model, and according to Gangal et al. (2007) is the usage of the fourth power error function rather focusing on minimizing the largest errors than minimizing the average error. When small errors are equally important as large errors, the mean absolute error might be preferred. The standard error function is the mean squared error, which is suitable for a large number of problems (Bishop, 2006; Haykin, 2009).

Several alterations of the provided error functions are provided in the literature. One example is the cross-entropy error function. Nielsen (2015) conclude that the error function J_{MSE} suffers from learning slowdown for certain evaluation points, and proposes the use of a cross-entropy error function which has a better learning rate than the mean of squared errors, and is defined as

$$J = -\frac{1}{N}\sum_{m=1}^{N}\Big[y_m \ln(\hat{y}_m) + (1 - y_m)\ln(1 - \hat{y}_m)\Big], \quad \hat{y}_m, y_m \in]0, 1[. \tag{3.6.2}$$

When using the backpropagation algorithm, the error function has to satisfy two properties. The first property is that it must be possible to write the error function as an average of cost functions for individual training examples. The second property is that

the error function must be a function of the outputs of the neural network. (Nielsen, 2015) The choice of error function will impact the expected value estimation of the output, as well as the computational efficiency of the training.

A common alteration of the MSE is the root mean squared error (RMSE) which is defined by Hu et al. (1999) as

$$RMSE = \sqrt{\frac{1}{T}\sum_{i_1}^{T}(y_t - \hat{y}_t)^2}, \quad RMSE = \text{Root Mean Square Error.} \tag{3.6.3}$$

Some error functions are proposed for evaluating neural network models, that does not have the same relevance for optimizing network weights, or hyper-parameters. They make the errors relative, and hence make it possible to compare between different contexts. Some of them are provided by Maasoumi and Racine (2002); Hu et al. (1999) as

$$MAPE = \frac{1}{N}\sum_{m=1}^{N}\left|\frac{y_m - \hat{y}_m}{y_m}\right|, \quad MAPE = \text{Mean Absolute Percentage Error} \tag{3.6.4}$$

$$MdAPE = Median\left(\left|\frac{y_m - \hat{y}_m}{y_m}\right| \cdot 100\right), \quad MdAPE = \text{Median Absolute Percentage Error} \tag{3.6.5}$$

$$SIGN = \frac{1}{N}\sum_{i=1}^{N} z_m, \quad \text{where} \quad z_m = \begin{cases} 1 & \text{if} \quad y_m \hat{y}_m > 0 \\ 0 & \text{otherwise} \end{cases}. \tag{3.6.6}$$

3.6.5 Problem structure

The network training tries to find the set of weights $\mathbf{W}^*$ that minimizes the error function. The point where the error function is minimized will have the property of $\nabla J(\mathbf{W}) = 0$. Points that satisfy this condition is called stationary points, and can be categorised as minima, maxima and saddle points. For a convex optimization problem no saddle points exits and there will be only one stationary point, either a minima or a maxima. Consequently, for the minima problem, the local minima will coincide with the global minimum of the problem. (Bishop, 2006)

For neural network problems there is typically a non-linear relationship between the error function and the weights, as is the case for the sum of squares differences error function described earlier. This means that one iteration of the backpropagation model will reach a local minimum, and that we will not be able to determine whether this is the global minimum or not. Generally, comparison of several local minimas is required to reach a sufficiently good solution of a non-convex optimization problem. (Bishop, 2006)

Solving a non-convex optimization problem

For non-convex optimization problems, there exists techniques to systematically find different local minimas in a try to find a solution that is "good enough". One of them is to run the optimization heuristic multiple times, with different initial conditions. By doing so, the objective function value for different local minimas can be compared, and consequently better solutions can be identified than if only one iteration of the optimization heuristic was run. (Lundgren et al., 2008) Bishop (2006) claim that this might also be necessary in neural network training. As such, a model for choosing initial solutions has to be defined, as well as a stopping criterion for when to stop the iterations. The model for picking initial conditions depend on the problem structure, and the stopping criterion may depend on the problem structure as well or other constraints as available time, fault tolerance etc.

3.6.6 Optimization of neural network weights

When optimizing, training, the networks weights, the class of the network will determine what training algorithm that can be used. When conducting training of multilayer perceptrons, i.e. a MLFFNN, the gradient descent, using backpropagation algorithm for determining the error function derivatives with respect to each weight, is popular, partly because of its simplicity but also because of its computational efficiency (Haykin, 2009; Bishop, 2006). In the literature, there is a slight confusion of terminology. Some researchers refer to the complete training algorithm as backpropagation training, whereas some researchers refer, correctly, to the process of calculating the derivatives of the error function with respect to each weight. (Bishop, 2006) Haykin (2009) explains that a neural network training algorithm consists of two major phases, the forward phase and the backward phase:

1. The forward phase refers to the forward propagation of the input signal $\mathbf{x}$ to the network, that after being processed by the neurodynamics, eventually produce an output signal $\mathbf{y}$ given a set of fixed weights $\mathbf{W}$.
2. The backward phase adjusts the weights in the network with the purpose of decreasing the error $J(\mathbf{W})$ between the generated output signal $\hat{\mathbf{y}}$ and the desired output $\mathbf{y}$

The two phases are alternated with the purpose of incrementally decreasing the produced total error of the network. Hence, the algorithm will at step $n+1$ update the weights $\mathbf{W}_{n+1}$ to improve the network and produce a smaller error $J(\cdot)$ than what was produced with $\mathbf{W}_n$ such that

$$J(\mathbf{W}_{n+1}) < J(\mathbf{W}_n). \tag{3.6.7}$$

The total error function $J(\mathbf{W})$ for a set of weights $\mathbf{W}$ is calculated as a sum of the errors for each training example $J_m(\mathbf{W})$, $m = 1, 2, \ldots, N_t$, defined as

$$J(\mathbf{W}) = \sum_{m=1}^{N} J_m(\mathbf{W}). \tag{3.6.8}$$

In the area of backpropagation, gradient descent is a widely used optimization heuristic to perform the update of weights between the step n and $n+1$ and eventually find the $\mathbf{W}^*$(Bishop, 2006). The general updating schema of an optimization algorithm is $\mathbf{W}_{n+1} = \mathbf{W}_n + \Delta\mathbf{W}_n$, however in the case with gradient descent, with a minimizing objective function, the updating schema for each weight for a signal from neuron i to a neuron j in layer l is

$$w_{ij}^{(l)}(n+1) = w_{ij}^{(l)}(n) - \eta_{ij}^{(l)} \frac{\partial J(\mathbf{W}_n)}{\partial w_{ij}^{(l)}(n)} \tag{3.6.9}$$

where the parameter $\eta_{ij}^{(l)} > 0$ is called the learning rate parameter. The corresponding vectorized updating schema is defined as

$$\mathbf{W}_{n+1} = \mathbf{W}_n - \boldsymbol{\eta} \odot \nabla J(\mathbf{W}_n) \tag{3.6.10}$$

where $\odot$ is the Hadamard product, i.e. elementwise multiplication. In general, two optimization methods exist for training a neural network. The first is called the batch optimization method, and the second is called the on-line optimization method. The gradient descent in (3.6.10) requires processing of the entire training set for every evaluation of $\nabla J(\mathbf{W}_n)$, a technique called batch optimization method. Bishop (2006) describes that the simple batch gradient descent is a poor algorithm for batch optimization problems, and that methods as conjugate gradient methods and quasi-Newton methods (see e.g. Lundgren et al. (2008); Haykin (2009)) are more appropriate. However, remembering (3.6.8), i.e. that the error function $J(\mathbf{W})$ is a sum of errors for each data point $J_m(\mathbf{W})$, we can instead update the weights based on single data points iteratively as

$$\mathbf{W}_{n+1} = \mathbf{W}_n - \boldsymbol{\eta} \odot \nabla J_m(\mathbf{W}_n) \tag{3.6.11}$$

which is a technique called on-line gradient descent (also called stochastic gradient descent and sequential gradient descent). This technique has proven to be useful for network training on large data sets. On-line methods are more efficient in handling data redundancies, and are also prone to avoiding stationary points of the optimization problem since stationary points for the individual data points might not coincide with stationary points for the whole training example set. (Bishop, 2006) One can also use an intermediate model, where a number of individual data points are batched together rather than all data points as in the batch method.

For sufficiently small $\boldsymbol{\eta}$ the error function will for every update of the weights decrease. If the learning parameter is set too high, the algorithm face the risk of not converging to a local optimum as it will step past the local optimum. If the learning parameter is set to a too small value, the algorithm will be very slow. (Tsai and Hsiao, 2003) claim that the learning parameter in general can be set to 0.9. Haykin (2009) claims that all neurons ideally should learn at the same rate, and hence the learning rate parameter should be determined accordingly. Considerable properties when determining the learning rate parameter is that the last layers have larger local gradient than the first

layers, and similarly neurons with more inputs have larger local gradient, and should consequently be assigned a smaller learning rate parameter. In LeCun et al. (1998) it is proposed that the learning rate parameter for a signal from neuron i in layer l to a neuron j in layer $l+1$ should be inversely proportional to the square root of inputs M_l to the neuron j, i.e.

$$\eta_{ij}^{(l)} = \frac{1}{\sqrt{M_l}}. \tag{3.6.12}$$

Adaptive learning rate parameters can be utilized, i.e. the learning rate parameters are adjusted during the course of the optimization. The adaptive learning rate is used to deal with the changes of magnitude of the gradient during the course of the optimization, as well as the need for fine-tuning the learning rate when the optimization algorithm is close to a local minimum. (LeCun et al., 1998)

The backpropagation technique plays a vital role for computing the $\nabla J(\mathbf{W})$. The derivative can not simply be calculated using an analytical expression, since the error function not is defined for intermediate neurons, only for the output layer. Bishop (2006) describes the backpropagation technique for an on-line method implementation, hence describing how the $\nabla J_m(\mathbf{W})$ is computed. Remembering that

$$a_j^{(l)} = \sum_{i=1}^{M_l} w_{ij}^{(l)} z_i^{(l)} \tag{3.6.13}$$

and

$$z_j^{(l+1)} = \varphi(a_j^{(l)}) \tag{3.6.14}$$

we note that J_m depends on $w_{ij}^{(l)}$ via the summed input $a_j^{(l)}$ to node j in the next layer. We can according to Persson and Böiers (2005) via the partial derivative chain rule write

$$\frac{\partial J_m}{\partial w_{ij}^{(l)}} = \frac{\partial J_m}{\partial a_j^{(l)}} \frac{\partial a_j^{(l)}}{\partial w_{ij}^{(l)}}. \tag{3.6.15}$$

From (3.6.13) we have that $\frac{\partial a_j^{(l)}}{\partial w_{ij}^{(l)}} = z_i^{(l)}$. Let us also denote $\frac{\partial J_m}{\partial a_j^{(l)}} = \delta_j^{(l)}$ as the error term. We now together with (3.6.16) have

$$\frac{\partial J_m}{\partial w_{ij}^{(l)}} = \delta_j^{(l)} z_i^{(l)}. \tag{3.6.16}$$

As such, we need to calculate $\delta_j^{(l)}$ for every neuron j in the output layer and the hidden layers to receive the total derivative. For the input layer, the error term is not defined, and will as such not be calculated.

The backpropagation algorithm will depend on the choice of neural network class, error function and choice of activation functions in both the hidden layers, but also the output layer. To exemplify how the backpropagation algorithm works in practice, we will consider a MLFFNN with a common choice of activation function in the output layer, i.e. the identity function $\varphi_o(a) = a$. The error function will be the sum of M_L squared errors defined as

$$J_m = \frac{1}{2}\sum_{j=1}^{M_L}(\hat{y}_{mj} - y_{mj})^2 \tag{3.6.17}$$

where $\hat{y}_{mj}$ and y_{mj} is the output of the neural network and the desired output respectively for output node $j \in \{1, 2, \ldots, M_L\}$ of training example $m \in \{1, 2, \ldots, N\}$. These choices give the derivative of the error function with respect to the weights $w_{ij}^{(L-1)}$ to the output layer

$$\frac{\partial J_m}{\partial w_{ij}^{(L-1)}} = (\hat{y}_{mj} - y_{mj}) z_i^{(L-1)}. \tag{3.6.18}$$

Hence, by comparing (3.6.16) and (3.6.18) the error term $\delta_j^{(L-1)}$ for the node j in the output layer L is the difference between the output signal $\hat{y}$ and the desired response y

$$\delta_j^{(L-1)} = \hat{y}_j - y_j. \tag{3.6.19}$$

For the M_l neurons in the hidden layers before layer L there is no specific desired output, the error term rather has to be determined recursively backwards. Note that layer l can be any layer, either the output layer, or a hidden layer. Again, utilizing the partial derivative chain rule we get

$$\delta_i^{(l)} = \frac{\partial J_m}{\partial a_i^{(l)}} = \sum_{j=1}^{M_{l+1}} \frac{\partial J_m}{\partial a_j^{(l+1)}} \frac{\partial a_j^{(l+1)}}{\partial a_i^{(l)}}. \tag{3.6.20}$$

We previously defined $\frac{\partial J_m}{\partial a_j^{(l+1)}} = \delta_j^{(l+1)}$. We also know from (3.6.13) and (3.6.14) that $a_j^{(l+1)} = \sum_i w_{ij}^{(l+1)} \varphi(a_i^{(l)})$, and conclusively we now end up with the backpropagation formula

$$\delta_i^{(l)} = \varphi'(a_i^{(l)}) \sum_{j=1}^{M_{l+1}} w_{ij}^{(l+1)} \delta_j^{(l+1)}. \tag{3.6.21}$$

As we have defined $\delta_j^{(L-1)}$ for the output layer, we can backpropagate the errors in the network with arbitrary number of layers. Bishop (2006) summarizes the backpropagation algorithm with the four steps i-iv below, and completed with a loop and a weight

update schema we receive the network weight optimization algorithm for the on-line training methodology.

1. While $|\nabla J(\mathbf{W}_n)| >$ threshold

    I For every training example $m = 1, 2, \ldots, N$ repeat

    i. Forward propagate a training example x_m to find all neurons' activations

    ii. Calculate $\delta_j^{(L-1)}$ for the output neurons

    iii. Backpropagate $\delta_j^{(L-1)}$ using the backpropagation formula to obtain $\delta_j^{(l)}$ for every neuron j in every hidden layer l

    iv. Calculate the derivatives using (3.6.16)

    v. Update the weights according to the optimization heuristic of choice

The procedure of one outer loop, i.e. I above, is usually called an epoch. An epoch is thus referring to forward and backward propagation of all the training examples. Usually several epochs are needed for the algorithm to converge to a stationary point.

Variants of the standard gradient descent optimization

One common modification of the gradient descent backpropagation training algorithm is the introduction of a momentum term. Kaastra and Boyd (1996) describe the addition of the momentum term to the gradient descent by changing the updating schema $\Delta\mathbf{W}_n$ to

$$\Delta\mathbf{W}_n = -\eta \odot \nabla J(\mathbf{W}_n) + \alpha\Delta\mathbf{W}_{n-1} \tag{3.6.22}$$

where α is the momentum term that assigns a weight to the previous weight update term $\mathbf{W}_{n-1}$. LeCun et al. (1998) describe a couple of classical second order optimization algorithms that utilize information in the hessian of the objective function: the Newton method, Conjugate Gradient method, Quasi-Newton (BFGS) method, Gauss-Newton and Levenberg Marquardt. These classical second order methods are according to LeCun et al. (1998) impractical in almost all useful cases, as methods using full hessian information only can be applied to very small networks trained in batch mode, and these networks are not in need of being improved as they are quickly computed. The authors further claim that a carefully tuned on-line gradient descent algorithm is hard to beat on large classification problems. For smaller functional approximation problems, the conjugate gradient is considered by LeCun et al. (1998) to be the best combination of speed, reliability and simplicity. As a consequence of these arguments, we will not describe any of the second order algorithms, nor how to backpropagate the hessian.

Weights initialization

In order to be able to run a search optimization heuristic, the initial set of weights has to be determined. (Lundgren et al., 2008) According to Ng (2016), it is important

to break the symmetry of the initial solution in the area of the backpropagation algorithm, meaning that the weights should be initialized to distinguished values. The implication of initializing the weights to the same number is that all the neurons (in a layer) will be calculating the same function of the input data, creating vast redundancies in the network. Also, given such an initialization the network will not be a universal approximator of functions. Ng (2016) rather suggest that each weight should be randomly initialized such that each element in $\mathbf{W} \in [-\epsilon, \epsilon]$. Haykin (2009) points at the fact that the choice of ϵ can impact the performance of the backpropagation algorithm since the initialization weights will greatly impact the training efficiency of the network. Too large weights will saturate the activation function, and to small weights will result in small gradients, slowing the learning down LeCun et al. (1998). There are numerous strategies for randomizing the initialization weights, Glorot and Bengio (2010) claim that a common strategy is to initialize the biases to 0 and the weights $w_{ij}^{(l)}$ from neuron i at layer l to neuron j at layer $l+1$ with M_l inputs to

$$w_{ij}^{(l)} \sim U\big[\frac{-1}{\sqrt{M_l}}, \frac{1}{\sqrt{M_l}}\big]. \tag{3.6.23}$$

Even though Bengio (2012) claim that the weight initialization for a neural network only has slight effect on the result, he propose different methods for increasing performance (finding better solutions). If computational power is at hand, he propose to run the optimization multiple times, 5 to 10 times, for a small set of parameter values. Another technique is model averaging (Section 3.4.4), such as Bagging (as explained by Breiman (1996)) and Bayesian methods. Briefly, model averaging is the process of averaging the outputs of multiple trained networks, where the different solutions depend on e.g. weight initialization, use of different input variables, use of different architectures, or use of different training examples (namely Bagging).

Another popular model for initializing the weights is the Nguyen and Widrow (1990) model. The general idea is to pick weights so that the active regions of the neurons in the hidden layers will be distributed approximately evenly over the input space to the neurons. Each hidden node will initially be assigned its own interval of the input space. If the transfer function outputs the interval $[-1, 1]$ and the inputs are preprocessed to $[-1, 1]$ the interval that has to be covered has the length 2. Nguyen and Widrow (1990) explain the methodology by considering a 2-layer neural network with one input x and H hidden neurons, where each hidden neuron is responsible for an interval of length $2/H$. The weight from the input to each hidden neuron is w_i and the bias weight is w_{b_i}. Consider a sigmoid transfer function that is approximately linear on

$$-1 < w_i x + w_{b_i} < 1 \qquad \Rightarrow \qquad -1/w_i - w_{b_i} < x < 1/w_i - w_{b_i} \tag{3.6.24}$$

has the length $2/w_i$. This means that $2/w_i = 2/H$ and $w_i = H$. Nguyen and Widrow (1990) use $w_i = 0.7H$ to get somewhat overlapping intervals. The bias weight w_{b_i} is picked so that the center of an interval is located at

$$x = -w_{b_i}/w_i \sim U[-1, 1] \qquad \Rightarrow \qquad w_{b_i} \sim U[-|w_i|, |w_i|]. \tag{3.6.25}$$

3.6.7 Generalization error

When training a neural network, it learns patterns of input-output combinations given by the training examples. The goal is to find a network that is able to generalize well, i.e. to be able to accurately predict an output even if the future input data is slightly different than the trained examples. However, a network that is provided with many training examples, or a network that model a very high order polynomial (many hidden layers and/or many hidden neurons), might risk having a low generalization ability, even if the functional fit to the training data is good. Figure 3.6.4 shows an example of poor generalization ability, as the output is not able to predict out-of-sample data, described by the true functional relationship in Figure 3.6.3, which generally never is known. This problem is known as overfitting or overtraining. (Haykin, 2009)

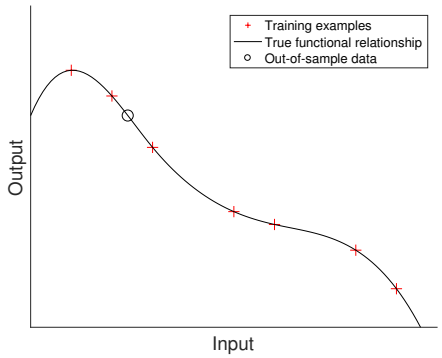

Figure 3.6.3: True functional relationship between input and output

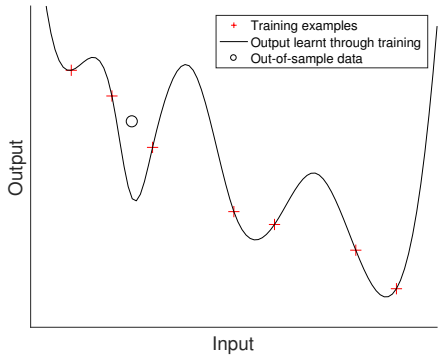

Figure 3.6.4: Trained network with poor generalization ability

Similarly, a network can underfit the data, which means that the model is too simple in regards to the data that it is trying to model, e.g. a linear model where the underlying relationship of the data is exponential. (Ng, 2016)

The conclusion from this example is that the training error is not a good metric for the generalization ability of a neural network, something that is supported by most researchers, e.g. Bishop (2006); Haykin (2009). Consequently, we need to formulate techniques in order to overcome this problem.

Prechelt (1998) state that overfitting can be avoided either by reducing the number of dimensions in the parameter space or by reducing the size of each dimension. The former can be done via greedy constructive learning, pruning, weights sharing, or as proposed by Ng (2016) a model selection algorithm. The latter can be done with regularization or early stopping. The two main options to resolve the issue of overfitting for a neural network is according to Ng (2016) the use of a model selection algorithm and/or regularization.

Regularization

The regularization approach refers to the introduction of a regularization term in the error function. According to Bishop (2006), the simplest regularizer is the sum of all

quadratic weights which incorporated to the error function gives the regularized error function

$$\tilde{J}(\mathbf{w}) = J(\mathbf{w}) + \frac{\phi}{2}\mathbf{w}^T\mathbf{w} \tag{3.6.26}$$

where $\phi \geq 0$ is the regularization coefficient. This regularization is in machine learning known as weight decay. The factor ϕ will control the level of over- or underfitting, and typically, a higher value of ϕ will decrease the magnitude of the weights. There are of course an infinite number of regularized error functions $\tilde{J}(\mathbf{w})$, and a general form is described by

$$\tilde{J}(\mathbf{w}) = J_D(\mathbf{w}) + \phi J_W(\mathbf{w}) \tag{3.6.27}$$

where ϕ controls the importance of the error deriving from the regularization term $J_W(\mathbf{w})$ and the data dependent error $J_D(\mathbf{w})$. (Bishop, 2006)

Early stopping

Another method for decreasing the risk of overfitting is the technique of early stopping. The technique is described by Prechelt (1998) and is based on the idea that as the training error decreases with the number of epochs, the generalization error will at some point increase as overfitting is starting to occur, see Figure 3.6.5. In theory, once the generalization error has increased for one epoch, the weights for the previous epoch should be used. In practice, the generalization error is usually approximated with the error of the validation set. Also in practice, the validation error usually contains several local minimas which implies that the process becomes a trade-off between finding a low generalization error and computation time, see Figure 3.6.6. As such, more sophisticated stopping criteria that result in a good "time-performance ratio" have to be defined, other than a criterion that stops when the generalization error has increased.

Prechelt (1998) proposes three classes of stopping criteria. Let us denote $J_{training}(t)$ as the average training set error, $J_{validation}(t)$ as the average validation set error, and $J_{test}(t)$ as the average test set error measured after epoch t. Now let us define

$$J_{optimal}(t) = \min_{t' \leq t} J_{validation}(t'). \tag{3.6.28}$$

The generalization loss GL at epoch t is defined as the percental increase in generalization error at epoch t with regards to the best generalization error

$$GL(t) = 100\left(\frac{J_{validation}(t)}{J_{optimal}(t)} - 1\right). \tag{3.6.29}$$

There might though be instances where the generalization error increases a lot during a part where the training error still decreases rapidly. In such a setting, Prechelt (1998) remark that the generalization error might decrease rapidly again, based on the idea

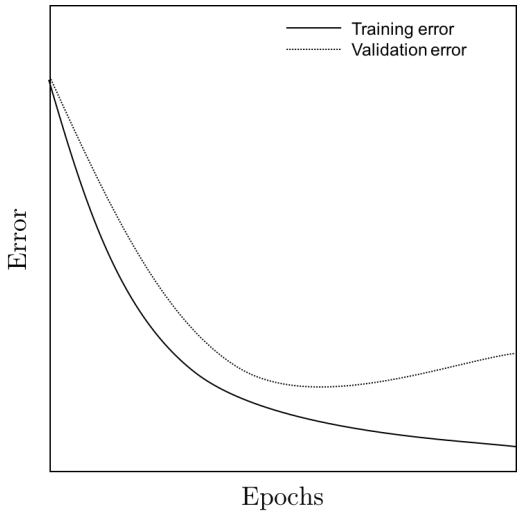

Figure 3.6.5: Theory of training vs. validation error for a neural network

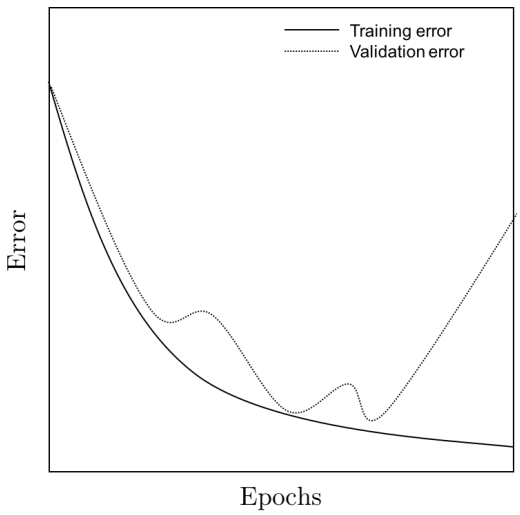

Figure 3.6.6: Common training vs. validation error characteristics

that overfitting starts to occur first when the training error decreases slowly. Prechelt (1998) therefore define a metric that measure how much the average training error during a training strip of length k was larger than the smallest training error during the same training strip. A training strip is a sequence of epochs $n, n+1, \ldots, n+k$. Let us now define

$$P_k(t) = 1000\left(\frac{\sum_{t'=t-k+1}^{t} J_{training}(t')}{k \cdot \min\limits_{t-k+1 \leq t' \leq t} J_{training}(t')} - 1\right). \tag{3.6.30}$$

With these definitions, we can now define the three classes of stopping criteria defined by Prechelt (1998), which are presented in Table 3.6.1.

Table 3.6.1: Three stopping criterias defined by Prechelt (1998)

Class	**Symbol**	**Criteria**
1	GL_α	Stop after first epoch t with $GL(t) > \alpha$
2	PQ_α	Stop after first end-of-strip epoch t with $\frac{GL(t)}{P_k(t)} > \alpha$
3	UP_s	Stop after epoch t iff UP_{s-1} Stops after epoch $t-k$ and $J_{validation}(t) > J_{validation}(t-k)$
	UP_1	Stop after first end-of-strip epoch t with $J_{validation}(t) > J_{validation}(t-k)$

The third class of stopping criteria in Table 3.6.1 stops when the generalization error has increased for s consecutive steps. This criterion measures the sign of the difference in generalization error between epochs rather than the size of the difference, and can therefore learn to recognize a pattern of consistency of generalization error increases.

As none of these stopping criteria might be terminated alone, Prechelt (1998) propose to use complementing stopping criteria, e.g. when progress of validation error drops below 0.1 or after a maximum of 3000 epochs.

By conducting tests with 14 different stopping criteria on 12 different learning tasks with 12 different network architectures, resulting in 1296 runs per learning algorithm, Prechelt (1998) came to three fundamental conclusions listed as

1. Unless small improvements of network performance is needed (e.g. 4 %) use fast stopping criteria (parameters α, s), or else time is increased severely ($\sim$4 for 4 % performance increase)
2. If high probability of finding a "good solution" is preferred, use GL criterion
3. If high average quality of solutions is preferred, use PQ criterion if network overfits only little, else use UP criterion

Whether or not these rules generalize to other learning tasks than those used in the paper is not validated though. (Prechelt, 1998)

3.6.8 Hyper-parameter optimization

As there are numerous hyper-parameters to decide upon when designing a neural network, we have to define a heuristic that selects the hyper-parameters $\boldsymbol{\lambda}^* \in \Lambda$ that generates the best generalization ability, thus the best network. This process is commonly called hyper-parameter optimization. As the objective function not is differentiable on the hyper-parameters, we must resort to other methods than gradient-based optimization models. The hyper-parameter optimization problem can mathematically be defined as

$$\boldsymbol{\lambda}^* = \arg\min_{\boldsymbol{\lambda} \in \Lambda} \Psi(\boldsymbol{\lambda}) \tag{3.6.31}$$

where $\Psi(\boldsymbol{\lambda})$ is the generalization error given the hyper-parameters $\boldsymbol{\lambda}$. The optimization problem is however in practice often approximated with

$$\boldsymbol{\lambda}^* \approx \arg\min_{\boldsymbol{\lambda} \in \{\boldsymbol{\lambda}^{(1)} \dots \boldsymbol{\lambda}^{(S)}\}} \Psi(\boldsymbol{\lambda}) \tag{3.6.32}$$

where the trial set $\{\boldsymbol{\lambda}^{(1)} \dots \boldsymbol{\lambda}^{(S)}\}$ consists of S trial points. Bergstra and Bengio (2012) conclude that the most commonly used strategy is a combination of grid search and manual search. The generalization error can be computed using a single validation set, or by using the technique of cross-validation, described later. The most critical step in hyper-parameter optimization is according to Bergstra and Bengio (2012) the choice of trial points. Let $L^{(k)}$ be values for a variable k in Λ that is promising, and manually chosen for the hyper-parameter optimization. In manual search, all of the trial points are chosen manually. In grid search, the cartesian product of manually identified values $\{L^{(1)} \dots L^{(k)}\}$ are chosen to be the trial set. Hence, $\Psi(\boldsymbol{\lambda})$ is evaluated for every pair in the cartesian product, and the set that minimized $\Psi(\boldsymbol{\lambda})$ is chosen to approximate $\boldsymbol{\lambda}^*$. This approach suffers from the curse of dimensionality, which means that the number of trial points to be evaluated grows exponentially with the number of hyper-parameters. In practice, this means that only a few discretized values for every hyper-parameter can be evaluated.

Bergstra and Bengio (2012) argue that random search of parameter values largely improves the optimal parameter search in high-dimensionality search spaces. The strategy draws parameter values from a uniform distribution over the same parameter space that would be spanned by the grid search, and evaluates these on $\Psi(\boldsymbol{\lambda})$. Likewise, the hyper-parameters that minimized $\Psi(\boldsymbol{\lambda})$ is chosen to approximate $\boldsymbol{\lambda}^*$. The advantage is explained in Figure 3.6.7, where optimization of two hyper-parameters are considered using grid search and random search respectively. Consider this situation where one wants to maximize $f(y, x) = h(y) + g(x)$. In this case, $f(y, x) \approx g(x)$, which is benamed that f has low effective dimension. In the case of grid search, $g(x)$ is only evaluated on three distinct values of x, and consequently misses to evaluate it at a point where x gives a high output. On the contrary, the random search evaluates $g(x)$ on nine distinct values, and consequently is able to find a better solution than the grid search using the same number of trial points. For the grid search, the trial points are more evenly distributed on the two-dimensional plane, but not when projected to the axes of the hyper-parameters. The random search has a slightly more uneven distribution in the plane, but is more evenly distributed when projected on the axes. (Bergstra and Bengio, 2012)

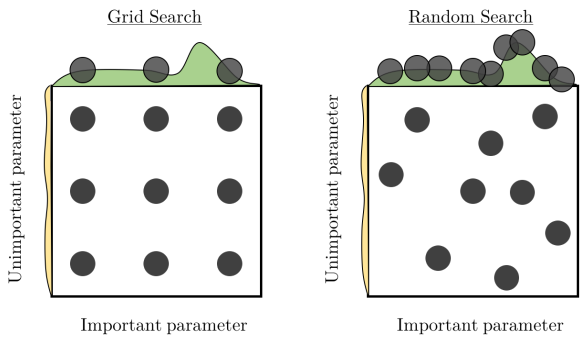

Figure 3.6.7: Grid and random search for hyper-parameters for nine trials respectively. Optimization of $f(y, x) = h(y) + g(x) \approx g(x)$. It is clear that grid search only evaluate $g(x)$ in three distinguished points, and misses the functional value top, whereas in random search $g(x)$ is evaluated in nine distinguished points and is better at hitting the functional value top, leading to a better hyper-parameter optimization in the same number of trials

The same principle is often the case in practical applications. If a neural network designer beforehand knows which hyper-parameters that will affect the final model the most, he could likely set up a decent grid search, however that is usually not the case. Some dimensions might be pruned, but usually there will still be uncertainties in a number of hyper-parameters. Generally, the random search model will make the hyper-parameter optimization more efficient. Bergstra and Bengio (2012) compare the efficiency of the random search to the grid search and find that 8 trials with random search match or outperform the performance of 100 grid searches on average.

Cross validation

When evaluating the generalization error, we have to approximate the out-of-sample data with something. As previously mentioned, this is usually done using a validation

data set. Bishop (2006) highlights the importance of having a sufficiently amount of data partitioned to the validation set. If data is plentiful, a portion of the data can simply be partitioned as the validation set. However, if the data is limited, this validation set will likely be small, implying a noisy generalization error estimate. By applying the K-fold cross validation technique the usage of the data is rationalized and the noise of the generalization error can be decreased. The network is evaluated K times using $\frac{K-1}{K}$ of the data for training and $\frac{1}{K}$ of the data for evaluation in every evaluation as illustrated in Figure 3.6.8. The generalization error of the specific trained network is then set to the average error over the evaluations. As such, all of the data is used for evaluating the performance of the network. This also provide a network validated on different types of data. The testing set is held separate of the cross validation process.

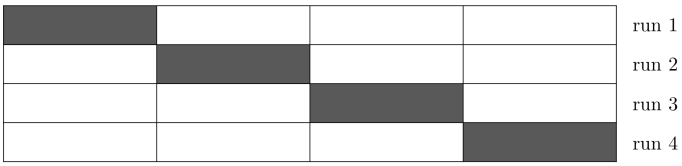

Figure 3.6.8: Illustration of a 4-fold cross validation. The training and validation set is partitioned into K (4 in this case) subsets and run K times where the validation set will be the gray subset in every run.

Early stopping criteria might be used together with the cross validation technique (Enke and Thawornwong, 2005). The K-fold cross validation technique will increase the number of training runs by a factor of K, which might be a disadvantage, especially in a setting with computationally expensive training. Further, if several hyper-parameters are to be distinguished, the number of training runs is increased exponentially to the number of hyper-parameters (Bishop, 2006).

3.7 Portfolio optimization

To validate whether the estimated expected returns from the neural network models are useful, i.e. if they can aid in generating excess return on the market, a trading strategy has to be defined. Numerous mathematical optimization models are defined in the litterature, e.g. mean variance (Amenc and Sourd, 2003) and two-stage stochastic programming without recourse decisions (Shapiro et al., 2009). The mathematical portfolio optimization models determine optimal asset allocation, or portfolio weights, at specific time steps. As such, this section introduces portfolio optimization models.

3.7.1 Utility functions

Utility functions define the utility $U(w_i)$ of a wealth w_i. If several outcomes are possible with probability p_i then the expected utility is defined as

$$E[U(W)] = \sum_{i=1}^{n} p_i U(w_i). \tag{3.7.1}$$

As such, utility functions can be used as to describe the risk aversion of an investor when the wealth is stochastic. The investor can be categorised into one of three risk categories according to

$$U(E[W]) - E[U(W)] = \begin{cases} > 0 \implies \text{risk averse} \\ = 0 \implies \text{risk neural} \\ < 0 \implies \text{risk seeking} \end{cases} \tag{3.7.2}$$

(Amenc and Sourd, 2003)

Luenberger (1998) describe that the utility function generally can take any form, as long as it is an increasing continuous function, i.e. for $w_1 > w_2$ it must hold that $U(w_1) > U(w_2)$. Further, Luenberger (1998) present the most common utility functions being the quadratic, power and exponential utility function.

The quadratic utility function is defined with risk aversion parameter a as

$$U(w) = w - aw^2, \quad a > 0, \quad w < \frac{1}{2a}. \tag{3.7.3}$$

The power utility function is defined with the risk aversion parameter γ, where $\gamma = 1$ is a risk neutral investor and $\gamma < 1$ is a risk averse investor, as

$$U(w) = \begin{cases} \ln(w) & \text{for } \gamma = 0 \\ \frac{w^\gamma}{\gamma} & \text{for } \gamma \leq 1, \gamma \neq 0 \end{cases}. \tag{3.7.4}$$

The exponential utility function is defined with risk aversion parameter b as

$$U(w) = -e^{-bw}, \quad b > 0. \tag{3.7.5}$$

Luenberger (1998) state that in the long run, an investor with a logarithmic utility function, $U(w) = \ln(w)$, will have higher wealth than an investor with different investment strategy with probability 1. Although, this is associated with high risk and the stochastic dominance might take very long time, e.g. several thousands of years. As such, in the short term, other investment strategies might be beneficial.

3.7.2 Mean Variance

One of the most common models for computing the optimal weights of a portfolio is the Mean Variance model defined by Markowitz (1952). In this model the returns are assumed to be normally distributed $N(\boldsymbol{\mu}, \mathbf{C})$ and the optimal portfolio weights are those that maximize the expected value of the portfolio's utility as

$$\max_{\mathbf{1}^T\mathbf{w}=1} \quad \boldsymbol{\mu}^T\mathbf{w} + \frac{\gamma_{MV} - 1}{2}\mathbf{w}^T\mathbf{C}\mathbf{w} \tag{3.7.6}$$

where γ_{MV} is a measure of the investor's risk aversion and $\mathbf{w}$ is the portfolio weights. For this model the investor has to have a quadratic utility function and only consider the first two moments (i.e. expected value and volatility). Further, there can be no market imperfections (taxes, transaction costs, etc.), interest on lendings and deposits has to be the same, and the holdings are unlimited. (Amenc and Sourd, 2003)

3.7.3 Stochastic Programming

In the area of mathematical optimization, stochastic programming is modelling of optimization problems that involve uncertainty, contrary to deterministic optimization problems. As stated previously, financial times series involve uncertainty, and is consequently a field where stochastic programming can be applied. In this section we will consider a two-stage stochastic programming model without recourse decisions. Shapiro et al. (2009) describe a general stochastic programming problem in the area of statistical inference as

$$\underset{x \in X \subset \mathbb{R}^n}{\text{minimize}} \quad f(x) = E[g(x, \xi)] \tag{3.7.7}$$

where ξ is a stochastic vector with pdf supported on a set $\Xi \subset \mathbb{R}^d$ and g is a known function that satisfy $g : X \times \Xi \to \mathbb{R}$. In the two-stage stochastic programming problem, $g(x, \xi)$ is given by the optimal value of the corresponding second-stage problem, i.e. the optimal x given the outcome ξ in the second stage. We assume that $f(x)$ is finite valued for every $x \in X$, which means that for for $x \in X$ $g(x, \xi)$ is finite valued almost surely for $\xi \in \Xi$.

In practice, the problem has to be discretized using a sample of N scenarios of the random vector ξ, i.e. $\xi_1, \xi_2, \ldots, \xi_N$. This sample might be generated using historical data of N observations or by simulating the future N scenarios using Monte Carlo simulation. Once the sample is generated, the problem in (3.7.7) can be approximated by averaging $g(x, \xi_i)$ for the scenarios $i = 1, 2, \ldots, N$, and we now have the deterministic equivalent of the stochastic problem described by

$$\underset{x \in X \subset \mathbb{R}^n}{\text{minimize}} \quad \hat{f}(x) = \frac{1}{N} \sum_{i=1}^{N} g(x, \xi_i) \tag{3.7.8}$$

where $\frac{1}{N} = p$ can be interpreted as the probability of each scenario incurring. The approximated problem in (3.7.8) is also known as the sample average approximation (SAA). Shapiro et al. (2009) assume that each ξ_j has the same marginal distribution, cdf, as the data vector ξ, and if each ξ_j are independent, they are said to be independently identically distributed (i.i.d.). We note that if ξ_j are i.i.d. the approximation $\hat{f}(x)$ is an unbiased estimator of $f(x)$ as

$$E[\hat{f}(x)] = E\left[\frac{1}{N} \sum_{i=1}^{N} g(x, \xi_i)\right] = \frac{1}{N} \sum_{i=1}^{N} E[g(x, \xi_i)] = \frac{1}{N} \sum_{i=1}^{N} E[g(x, \xi)] = E[g(x, \xi)] = f(x) \tag{3.7.9}$$

with an error that decreases with $\sqrt{N}$ as

$$Var[\hat{f}(x)] = Var\left[\frac{1}{N}\sum_{i=1}^{N} g(x,\xi)\right] = \frac{1}{N^2}\sum_{i=1}^{N} Var[g(x,\xi)] = \frac{1}{N}Var[g(x,\xi)]. \quad (3.7.10)$$

As $N \to \infty$ the SAA problem $\hat{f}(x)$ converges to the real problem $f(x)$ with probability 1 under some regularity conditions. (Shapiro et al., 2009)

3.7.4 Scenario generation

Scenarios can according to Shapiro et al. (2009) be generated by using historical data or by performing Monte Carlo simulation. Monte Carlo simulation is stochastic by nature, as it is a random sampling of a variable from a probability distribution. Usually a random variate is drawn from a uniformly distributed variable, $X \sim U[0,1]$.

In the context of portfolio optimization, we have a set of random variables with correlation, simulating the portfolio hence must be done with regards to the correlation structure between the underlying random variables. Therefore, we can not independently draw random variates from $U(0,1)$, to generate scenarios using the Monte Carlo method we need to simulate a multivariate distribution, and as such we also need to estimate the multivariate distribution of the underlying assets. A copula is a function that specifies the correlation structure between random variables, which along with the univariate marginal distributions enables a description of the multivariate distribution. The technique is described in Cherubini et al. (2004); Trivedi and Zimmer (2005). The principle of Monte Carlo simulation using a copula is based on the inversion principle (Devroye, 1986) and the therorem by Sklar (1959).

Theorem 3.7.1 (Inversion principle) *Let F be a continuous distribution on R with inverse F^{-1} defined by*

$$F^{-1}(u) = \inf\{x : F(x) = u, 0 < u < 1\}.$$

If U is a uniform $[0,1]$ random variable, then $F^{-1}(U)$ has distribution function F. Also, if X has distribution function F, then $F(X)$ is uniformly distributed on $[0,1]$.

A consequence of Theorem 3.7.1 is that random variates with an arbitrary distribution function can be generated with uniformly distributed random variables. Also, random variates with an arbitrary distribution function can be transformed to uniformly distributed random variates. (Devroye, 1986)

Theorem 3.7.2 (Sklar's therorem) *Let $F \in F(F_1, \ldots, F_n)$ be an n-dimensional distribution function with marginal distributions $F_1, \ldots, F_n$. Then there exists a copula C (i.e. an n-dimensional distribution function with uniform marginals) such that*

$$F(x_1, \ldots, x_n) = C(F_1(x_1), \ldots, F_n(x_n))$$

Given univariate marginal distribution $F_i(x_i)$ we can according to Theorem 3.7.1 generate uniformly distributed random variate $u_i = F_i(x_i)$, and equivalently $x_i = F_i^{-1}(u_i)$. Hence the multivariate distribution function $F(x_1, \ldots, x_n)$ can be rewritten as $F(F_1^{-1}(u_1), \ldots, F_n^{-1}(u_n))$. The copula is now defined as

$$C(u_1, \ldots, u_n) \equiv F(F_1^{-1}(u_1), \ldots, F_n^{-1}(u_n)) = F(x_1, \ldots, x_n) \tag{3.7.11}$$

which means that the copula only manages the correlation, and the univariate properties are managed by $F_i(x_i)$. The copula C can be interpreted as the distribution function for multivariate uniformly distributed random variables computed by

$$\begin{aligned} C(u_1, \ldots, u_n) &= \int_{-\infty}^{u_1} \cdots \int_{-\infty}^{u_n} c(v_1, \ldots, v_n) dv_n \cdots dv_1 \\ &\overset{u_i = F_i(x_i)}{=} \int_{-\infty}^{F(x_1)} \cdots \int_{-\infty}^{F(x_n)} c(v_1, \ldots, v_n) dv_n \cdots dv_1 \\ &= \Big/ \text{subs. } v_i = F_i(s_i), \quad \frac{dv_i}{ds_i} = \frac{d(F_i(s_i))}{ds_i} = f_i(s_i) \Leftrightarrow dv_i = f_i(s_i) ds_i \Big/ = \\ &= \int_{-\infty}^{x_1} \cdots \int_{-\infty}^{x_n} c(F(s_1), \ldots, F(s_n)) \prod_{i=1}^{n} f_i(s_i) ds_n \cdots ds_1. \end{aligned} \tag{3.7.12}$$

We can also use (3.7.11) and write

$$C(u_1, \ldots, u_n) = F(x_1, \ldots, x_n) = \int_{-\infty}^{x_1} \cdots \int_{-\infty}^{x_n} f(s_1, \ldots, s_n) ds_n \cdots ds_1 \tag{3.7.13}$$

and by comparing (3.7.12) and (3.7.13) we identify that

$$f(x_1, \ldots, x_n) = c(F_1(x_1), \ldots, F_n(x_n)) \prod_{i=1}^{n} f_i(x_i). \tag{3.7.14}$$

The density function $f(x_1, \ldots, x_n)$ can be approximated using MLE, as described in Section 3.2.2, for observations $t = 1, \ldots, T$ with the loglikelihood function as

$$\begin{aligned} \ln(L) &= \ln\left(\prod_{t=1}^{T} f(x_t) \right) = \ln\left(\prod_{t=1}^{T} c(F_1(x_{t,1}), \ldots, F_n(x_{t,n})) \prod_{i=1}^{n} f_i(x_{t,i}) \right) \\ &= \sum_{t=1}^{T} \ln c(F_1(x_{t,1}), \ldots, F_n(x_{t,n})) + \sum_{i=1}^{n} \sum_{t=1}^{T} \ln f_i(x_{t,i}) \end{aligned} \tag{3.7.15}$$

which is a large optimization problem. By using the method *inference for margins* the optimization problem is solved approximately by estimating the univariate marginal distributions and the copula separately. In practice, the copula is usually estimated

from a set of different standard copula functions. The copula function that yields the highest log-likelihood value can be used as an estimation of the copula. Common multivariate copula functions used are the gaussian copula and the student's t copula. The gaussian copula function is defined as

$$c(\mathbf{u}) = \frac{1}{\sqrt{|\mathbf{P}|}} e^{-\frac{1}{2}\mathbf{N}^{-1}(\mathbf{u})^T(\mathbf{P}^{-1}-\mathbf{I})\mathbf{N}^{-1}(\mathbf{u})} \tag{3.7.16}$$

where $\mathbf{N}^{-1}(\mathbf{u}) = (N^{-1}(u_1), \ldots, N^{-1}(u_n))^T$, N is the cumulative distribution function (CDF) of a standard normally distributed random variable, $u_i = F_i(x_i)$, $\mathbf{P}$ is the correlation matrix and $|\cdot|$ is the determinant. Using MLE, $\mathbf{P}$ is the parameter that has to be estimated.

The student's t copula is defined as

$$c(\mathbf{u}) = \frac{\left(\Gamma\left(\frac{v}{2}\right)\right)^{n-1}\left(\Gamma\left(\frac{v+n}{2}\right)\left(1+\frac{\mathbf{\Phi}^{-1}(\mathbf{u})^T\mathbf{P}^{-1}\mathbf{\Phi}^{-1}(\mathbf{u})}{v}\right)\right)^{-\frac{v+n}{2}}}{\left(\Gamma\left(\frac{v+n}{2}\right)\right)^{n}\sqrt{|\mathbf{P}|}\prod_{i=1}^{n}\left(1+\frac{1}{v}\left(\Phi^{-1}(u_i)\right)^2\right)^{-\frac{v+1}{2}}} \tag{3.7.17}$$

where $\mathbf{\Phi}^{-1}(\mathbf{u}) = (\Phi^{-1}(u_1), \ldots, \Phi^{-1}(u_n))$, Φ is the CDF of a student's t distributed random variable, $u_i = F_i(x_i)$, $\mathbf{P}$ is the correlation matrix, $|\cdot|$ is the determinant, v is the degrees of freedom, and $\Gamma(t) = \int_0^\infty x^{t-1}e^{-x}dx$. Using MLE, $\mathbf{P}$ and v are the parameters that have to be estimated.

A consequence of using the MLE as the estimator of the parameters in the copula functions, is that the technique assumes that the parameters are stationary. By conducting MLE for the gaussian and student's t copula, the correlation matrix is held constant for the entire sample. Instead, estimating a time varying correlation matrix, one can use multivariate GARCH described in Appendix C.2.

In order to keep the correlation structure of the copula function when doing the Monte Carlo simulation, Cholesky decomposition might be used. As the correlation matrix by definition is semi-definite (remember $\mathbf{w}^T\mathbf{C}\mathbf{w} \geq 0$), the decomposition $\mathbf{P} = \mathbf{L}\mathbf{L}^T$ is possible, where $\mathbf{L}$ is a lower triangular matrix. Now, consider the simulation of $\mathbf{Z} \sim N(0, \mathbf{I})$ that with the transformation $\mathbf{a} = \mathbf{L}\mathbf{Z}$ gives

$$E[\mathbf{a}\mathbf{a}^T] = E[\mathbf{L}\mathbf{Z}\mathbf{L}^T\mathbf{Z}^T] = \mathbf{L}E[\mathbf{Z}\mathbf{Z}^T]\mathbf{L}^T = \mathbf{L}\mathbf{I}\mathbf{L}^T = \mathbf{P} \tag{3.7.18}$$

i.e. the original correlation matrix. So by simulating the vector $\mathbf{Z} = (Z_1, \ldots, Z_n)$ we can get $\mathbf{a} = (a_1, \ldots, a_n) = LZ$. To get uniformly distributed random variates that has the original correlation structure we transform $\mathbf{a}$ via the inversion principle

$$\mathbf{u} = (u_1, \ldots, u_n) = (N(a_1), \ldots, N(a_n)). \tag{3.7.19}$$

Finally, to get random variates x_i for the random variables of interest X_i, the inversion principle can be used with the univariate distributions F_i to get

$$\mathbf{x} = (x_1, \ldots, x_n) = (F_1^{-1}(u_1), \ldots, F_n^{-1}(u_n)). \tag{3.7.20}$$

Based on the relationship in (3.7.20) the univariate distributions have to be estimated in order to generate portfolio scenarios. Three different univariate distributions that is applicable in finance is described in Appendix D.

3.8 Evaluation

To be able to validate whether the purpose of the thesis is fulfilled, we need to evaluate the predictions of the expected returns and the portfolio optimization. This section describes models to evaluate the areas respectively.

3.8.1 Prediction of expected return

For predictions, both error metrics and statistical tests can be utilized to evaluate the performance. The most common statistical tests for forecast accuracy equality will be presented. The most promising tests will be described in depth, whereas the other will be presented briefly based on their large scope.

The error metrics in Section 3.6.4 can give a hint of the predictive performance of a forecast. Diebold (2015) claim that when comparing the predictive accuracy of different forecasts, a forecaster usually compare a number of forecasts with a number of outcomes on the predicted variable. Let $\epsilon_t = y_t - y_t^{(A)}$ denote the forecast error of forecast A at time t. A loss function $L(\cdot)$ is defined to quantify the error of the forecast, and might be e.g. the absolute forecast error, or the square of the forecast error, defined respectively by

$$L_1(\epsilon_t) = |\epsilon_t|, \quad L_2(\epsilon_t) = \epsilon_t^2. \tag{3.8.1}$$

However, the actual difference is not enough to tell whether one forecast is statistically better than another. Consider a comparison between two forecasts, $y_f^{(A)}$ and $y_f^{(B)}$, where the root mean of the quadratic losses (RMSE) are $\hat{RMSE}_A = 1.2$ and $\hat{RMSE}_B = 1.5$. Obviously $\hat{RMSE}_A < \hat{RMSE}_B$ and one might draw the conclusion that A provides better forecasts than B. Diebold (2015) describes that forecasting literature is filled with such conclusions, although, the superiority of forecast A might be based on merely luck of the actual values in the sample, thus the conclusion is statistically unsatisfactory.

Simple F-test

A simple statistical test of forecast equality is the simple F-test. Consider a hypothesis test with null hypothesis that the forecast error of two forecasts are equal. If the assumptions of two forecasts in Table 3.8.1 are satisfied

Table 3.8.1: Common forecast assumptions

Common forecast assumptions
1. Loss is quadratic
2. Forecast errors are:
(a) Zero mean
(b) Gaussian
(c) Serially uncorrelated
(d) Contermporaneously uncorrelated

then under the null hypothesis of equal forecast accuracy the forecast error variance is equal and the ratio of sample variances has the $F(T,T)$ distribution. Hence, under the null hypothesis it holds that

$$F = \frac{\frac{\epsilon_i^T \epsilon_i}{T}}{\frac{\epsilon_j^T \epsilon_j}{T}} = \frac{\epsilon_i^T \epsilon_i}{\epsilon_j^T \epsilon_j} \sim F(T,T) \tag{3.8.2}$$

where $\epsilon_x \in \mathbb{R}^{T\times 1}$, $x = i, j$, contains the errors of forecasts. However, due to the strict restrictions in Table 3.8.1, the test static in (3.8.2) is rarely useful. (Diebold and Mariano, 1995)

Granger-Newbold and Meese-Rogoff Tests

Granger and Newbold (1977); Meese and Rogoff (1988) have defined statistical tests of forecast accuracy equality between two forecasts. Both of these tests relaxes assumptions made in Table 3.8.1, but none of the model relaxes all of the assumptions at once. The Meese and Rogoff (1988) tests can relatively easy relax (2a)-(2c) one at a time, while keeping (2d) relaxed for all tests. Simulataneoues relaxation of (2a)-(2d) is possible, but very tedious (Diebold and Mariano, 1995). Granger and Newbold (1977) have similar properties for relaxing the assumptions, where simultaneous relaxation is tedious, and like the Meese and Rogoff (1988), assumption (1) may not be relaxed.

Diebold-Mariano Test

Diebold and Mariano (1995) defined a statistical test for forecast accuracy equality between two forecasts. The test can be adopted for a vast range of forecast situations, as it can handle non-quadratic and asymmetric loss functions, and the forecast errors can be non-gaussian, nonzero mean, serially correlated and contemporaneous correlated, i.e. deals with all assumptions in Table 3.8.1. Let $L(\epsilon_t)$ denote the loss associated with error ϵ. Let $d_t = L(\epsilon_t^{(1)}) - L(\epsilon_t^{(2)})$ denote the loss differential between forecasts 1 and 2. Following assumptions about the loss differential is made, equaling an assumption about stationarity of the loss differential:

$$\text{Assumptions: } \begin{cases} E[d_t] = \mu, \quad \forall t \\ Cov[d_t, d_{t-\tau}] = \gamma(\tau), \quad \forall t \\ 0 < Var[d_t] = \sigma_d^2 < \infty \end{cases} \tag{3.8.3}$$

The null hypothesis H_0 and the alternative hypothesis H_1 is now defined as (3.8.4). The hypotheses in (3.8.4) test for nonzero loss differential, i.e. that one forecast has lower loss than the other, irrespectively if it is forecast 1 or forecast 2. Alternatively, if we want to test that forecast 2 has higher accuracy than forecast 1 we might alter the hypotheses to a one-sided test as in (3.8.5).

Table 3.8.2: Null respectively alternative hypotheses for two- respectively two-sided test of loss differential nonzero property

Two-sided test		**One-sided test**	
$H_0 : E[d_t] = 0$ $H_1 : E[d_t] \neq 0$	(3.8.4)	$H_0 : E[d_t] = 0$ $H_1 : E[d_t] > 0$	(3.8.5)

Under H_0 and the central limit theorem (Blom et al., 2005) it must hold that the test statistic, DM, is asymptotically normally distributed as

$$DM = \frac{\bar{d}}{\hat{\sigma}_{\bar{d}}} \approx N(0,1) \tag{3.8.6}$$

where $\bar{d} = \frac{1}{T}\sum_{t=1}^{T} d_t$ is the sample mean of the loss differential and $\hat{\sigma}_{\bar{d}} = \sqrt{\frac{2\pi \hat{f}_d(0)}{T}}$ is a consistent estimate of the standard deviation of $\bar{d}$, where $f_d(0) = \frac{1}{2\pi}\sum_{-\infty}^{\infty} \gamma_d(\tau)$ and the autocovariance at lag τ is $\gamma_d(\tau) = E[(d_t - \mu)(d_{t-\tau} - \mu)]$. Suppose we have a $h > 1$ −step ahead forecast we can now obtain a consistent estimate of $2\pi f_d(0)$ as a weighted sum of the available sample autocovariances as

$$2\pi \hat{f}_d(0) = \sum_{\tau=-(T-1)}^{T-1} I\big(\frac{\tau}{h-1}\big)\hat{\gamma}_d(\tau) \tag{3.8.7}$$

where

$$\hat{\gamma}_d(\tau) = \frac{1}{T}\sum_{t=|\tau|+1}^{T} (d_t - \bar{d})(d_{t-\tau} - \bar{d}) \tag{3.8.8}$$

and

$$I\big(\frac{\tau}{h-1}\big) = \begin{cases} 1 & \text{when} \quad \big|\frac{\tau}{h-1}\big| \leq 1 \\ 0 & \text{otherwise} \end{cases} \quad . \tag{3.8.9}$$

We also note that

$$\hat{\gamma_d}(\tau) = \hat{\gamma_d}(-\tau), \quad \text{and} \quad I\big(\frac{\tau}{h-1}\big) = 0 \quad \text{for} \quad |\tau| > h-1 \tag{3.8.10}$$

and thus we can simplify (3.8.11) to

$$2\pi \hat{f}_d(0) = \hat{\gamma_d}(0) + 2\sum_{\tau=1}^{h-1} \hat{\gamma_d}(\tau). \tag{3.8.11}$$

For $h = 1$ we can further simplify the relationship in (3.8.11) to

$$2\pi \hat{f}_d(0) = \hat{\gamma_d}(0) \overset{(3.8.8)}{=} \frac{1}{T}\sum_{t=1}^{T}(d_t - \bar{d})^2. \tag{3.8.12}$$

Conclusively, if the test static DM violates the critical limit for the normal distribution, the null hypothesis can be rejected in favor of the alternative hypothesis. For example, if a two-sided test is conducted at a 95 % significance level, the null hypothesis will be rejected if $|DM| > z_{0.025} = 1.96$.

Pesaran-Timmermann Test

Further, Pesaran and Timmermann (1992) define a model-free test for evaluating the forecast accuracy in terms of the direction of the change in a variable of interest. The test gives information whether the forecast is statistically significantly predicting the correct sign of the change, and might hence be useful when the direction of the forecast is of interest, i.e. a forecast predicting if the change is positive or negative ignoring the magnitude of the change. Let y_t denote the actual change and $\hat{y}_t$ be the forecasted change for a series of length T. Let the null hypothesis be that $\hat{y}_t$ is not able to predict the sign of y_t, then under the null hypothesis it must hold that

$$PT = \frac{\hat{P} - \hat{P}_*}{\sqrt{Var[\hat{P}] - Var[\hat{P}_*]}} \sim N(0,1) \tag{3.8.13}$$

where the location values are estimated by

$$\hat{P} = \frac{1}{T}\sum_{t=1}^{T} I(y_t \hat{y}_t), \quad \text{and} \quad \hat{P}_* = \hat{P}_y \hat{P}_{\hat{y}} + (1 - \hat{P}_y)(1 - \hat{P}_{\hat{y}}) \tag{3.8.14}$$

and the variance is

$$Var[\hat{P}] = \frac{1}{T}\hat{P}_*(1-\hat{P}_*) \tag{3.8.15}$$

and

$$Var[\hat{P}_*] = \frac{1}{T}(2\hat{P}_y - 1)^2\hat{P}_{\hat{y}}(1-\hat{P}_{\hat{y}}) + \frac{1}{T}(2\hat{P}_{\hat{y}} - 1)^2\hat{P}_y(1-\hat{P}_y) + \frac{4}{T}\hat{P}_y\hat{P}_{\hat{y}}(1-\hat{P}_y)(1-\hat{P}_{\hat{y}}) \tag{3.8.16}$$

$$\hat{P}_y = \frac{1}{T}\sum_{t=1}^{T} I(y_t), \quad \hat{P}_{\hat{y}} = \frac{1}{T}\sum_{t=1}^{T} I(\hat{y}_t) \tag{3.8.17}$$

$$I(\cdot) = \begin{cases} 1 & \text{when} \quad \cdot > 0 \\ 0 & \text{otherwise} \end{cases} . \tag{3.8.18}$$

Then the null hypothesis is rejected in favor of the alternative hypothesis that $\hat{y}_t$ is able to predict the sign of y_t when $|PT| > z_{\frac{\alpha}{2}}$ where $z_{\frac{\alpha}{2}}$ is the critical limit at significance level α.

3.8.2 Portfolio optimization

When evaluating the performance of a portfolio it is not enough to only consider the return, the risk taken should also be emphasized in the performance evaluation. MPT and CAPM do provide a quantitative link between return and risk, and hence can provide a risk-adjusted return which enables comparison of portfolios with different risk levels. (Amenc and Sourd, 2003)

A common performance metric is the sharpe ratio defined by Sharpe (1966) that relates the portfolio return r_P less the riskfree rate r_f with the total risk of the portfolio, measured by the standard deviation σ_P. As the relationship is drawn from MPT and not CAPM, the metric does not inherit the citicism by Roll (1977). The Sharpe ratio S_P is defined as

$$S_P = \frac{r_P - r_f}{\sigma_P}. \tag{3.8.19}$$

Jensen (1967) defined a performance metric stemming from CAPM that evaluate the portfolio in relation to its systematic risk, β_P. The metric α_P is determined by linear regression of the model with the serially independent error term ϵ_t, $E[\epsilon_t] = 0$, defined as

$$r_{P,t} - r_{f,t} = \alpha_P + \beta_P(r_{M,t} - r_{f,t}) + \epsilon_t \tag{3.8.20}$$

and can be interpreted as the excess return of the portfolio on the market index. As the metric benchmarks against an index, it will not give a risk-adjusted return, and should hence be used when comparing portfolios with the same risk level. By the

nature of the linear regression model, a hypothesis test for the sign of α_P can be established, providing the significance level on possible excess return. The hypotheses are $H_0 : \alpha_P = 0$ and $H_1 : \alpha_P > 0$. Hence, the probability for the excess return deriving from pure chance can be determined. Commonly, a maximum probability of 5 % of pure chance (or likewise minimum 95 % significance level) is used to say that the excess return is statistically significant, which corresponds to a t-statistic for α_P of approximately 1.65[8].

Treynor (1965) defined the Treynor Index as a relationship between the portfolio return r_P less the riskfree rate r_f divided by its systematic risk β_P, described as

$$T_P = \frac{r_P - r_f}{\beta_P}. \tag{3.8.21}$$

The metric gives a systematic risk-adjusted return, and is hence best used when a portfolio is only a subset of the investors total holdings, as an investor that only hold portfolio P will still be exposed to unsystematic risk. A portfolio that is sufficiently risk-rewarded satisfies

$$T_P \geq r_M - r_f \tag{3.8.22}$$

as $r_M - r_f$ is the Treynor Index for the market portfolio as $\beta_M = 1$. When similarity applies, (3.8.22) describes the relationship between the Treynor Index and CAPM. (Amenc and Sourd, 2003)

3.9 Empirical tests

In this section we will present some empirical studies where researches have applied commonly used paradigm's of neural networks to predict financial time series of assets relevant to the context of this thesis. Further, their parameter choices will be provided and their conclusions will be highlighted in order to strengthen our choice of methodology.

3.9.1 The use of data mining and neural networks for forecasting stock market returns

Enke and Thawornwong (2005) examine different neural network classes to predict S&P500 less the riskfree rate (approximated with T-bill) on monthly basis. Specifically, FFNNs were developed to both predict the continuous value (regression) and the next period direction (classification). Two more network classes, one for each task, Generalized regression neural network (GRNN) and Probabilistic neural network (PNN) were also tested and all of them benchmarked against the tradition linear regression model. Due to the fact that the task of the network in this thesis is of a regression type, only their final choice of Neural Network for the regression task and

[8]One sided confidence interval, $T^{-1}(0.95) = 1.65$ with large number of degrees of freeedom. Corresponding number for a double-sided confidence interval is $T^{-1}(0.975) = 1.96$.

its parameter choices as well as the method utilized to determine them is provided in Table 3.9.1.

Table 3.9.1: Parameter choice of the optimal ANNs in Enke and Thawornwong (2005) research paper

Parameter type	**Method**	**Parameter choice**
Data period	Monthly basis	1976-1999
Input type	Knowledge based	Fundamental data
# Input	Knowledge based	31
# Input after RA[9]	Inductive learning decision tree	15
Preprocessing	Normalization	[-1,1]
Activation function	Previous studies	Sigmoid hyperbolic tangen function
Error function	Previous Studies	MSE
Weight optimization	Previous studies	Resilient Backpropagation[10]
Generalization improvement	Previous studies	Five-fold CV and ES
Hidden layers	Previous studies	1
Ouput	Problem dependent	1
Hidden neurons	Trial and error	21
Learning rate	Trial and error	0.2
Network selection criterion	Previous studies	RMSE

With a hypothesis of getting further improvement of the forecasting performance, Enke and Thawornwong (2005) also examined a portfolio network model, i.e. ensemble averaging, consisting of a 5-network-combination of the network architectures that produced the lowest RMSE in each omitted fold from the cross-validation experiment.

The evaluation was based on metrics of the performance on out-of-sample data. The metrics were: RMSE, Pearson Correlation between the actual out-of-sample return and the prediction, and the proportion of time the sign of stock return was correct. The results are provided in Table 3.9.2

Table 3.9.2: Performance metrics for the three different ANN models

Model	**RMSE**	**Correlation**	**Sign**
Portfolio Neural Network	1.1206	0.0528	0.6860
Original Neural Network	1.1614	0.0231	0.6628
Linear Regression	1.4467	0.0300	0.4767

This shows that utilizing ensemble averaging improved the FFNN with respect to all the metrics. Further, both of them shows a better estimation than the traditional linear regression model.

[9]Relevance Analysis

[10]A tweak of the gradient descent, where the magnitude of the gradient is discarded, and the magnitude of the weight update is calculated with a learning rate that increases in magnitude if the sign is the same for successive steps

Enke and Thawornwong (2005) highlight the success of improving the generalization ability of feed-forward neural networks by utilizing a combination of n-fold cross-validation and early stopping techniques. The method clearly helped improving the out-of-sample forecasts. Further, they express that including a combination of both technical and fundamental information as input during the relevance analysis would provide invaluable information that with no doubt would be a major improvement.

A simple trading strategy is also implemented to test if the predictions could generate any profit. The trading strategy is specified as such that the portfolio will invest all wealth in the stock index if the prediction of the return less the riskfree rate is positve, else it will invest everything in a T-bill. The portfolio is re-allocated monthly. The best level ANN, i.e. the portfolio ANN, did slightly beat a buy-and-hold strategy during the evaluated period, Nov-92 to Dec 99, with an average monthly return of 1.58 % and Sharpe ratio of 0.34, compared to a monthly return of 1.54 %.

3.9.2 An investigation of model selection criteria for neural network time series forecasting

Qi and Zhang (2001) investigate the potential of applying ANNs to forecast financial time series on a longer time horizon viz. 1 year, 4 years and 8 years. Further, they look at 3 different financial time series: S&P500, US interest rate (T-bill) and exchange rate between British pound and us Dollar (USD/BP). They utilize data points on a different horizon than the forecast horizon, monthly for S&P 500 and T-bill, and weekly for exchange rate. Special focus in this paper is to determine whether an in-sample model selection criteria is useful as a model selection approach versus cross-validation, referred to as an out-of-sample model due to the network evaluation in the validation set.

With motivation that the universal approximation theorem holds they construct a 3-layer FFNN and evaluate the performance while changing the number of input nodes used as well as the number of hidden neurons between 1-5. To evaluate the performance they look at several performance measures RMSE, MAE, MAPE, DA and Sign, but put extra focus on RMSE, the network with the lowest RMSE forecasting on 1 year horizon is presented in Table 3.9.3.

As benchmark they construct a linear auto regression (AR) model and a random walk (RW) model which are used on the S&P 500 data. Qi and Zhang (2001) claim that both models are according to the literature relevant and applicable when predicting the return of financial assets. The performance provided in Table 3.9.3 shows the relevance of using ANN to predict estimated return on all the time horizons.

Qi and Zhang (2001) draw the conclusion that the popular in-sample criteria AIC and BIC are not very useful in neural network time series forecasting. One reason is that the criteria penalize models with more parameters which is useful in statistics where the number of parameters often is small whereas time series prediction and neural network in particular tends to involve more parameter in order to do an adequate fit. Further, the results of the study shows that there is no connection between the model

[11] Backpropagation of gradient

[12] AIC=Akike Information Criterion, BIC=Bayesian Information Criterion

Table 3.9.3: Parameter choice of the optimal ANNs in Qi and Zhang (2001) research paper

Parameter type	**Method**	**Parameter choice**
Data period	Monthly basis	1954-1992
Input type	Previous studies	Past returns
# Input	Trial and Error	5
Activation function (HL)	Previous studies	Sigmoid
Activation function (Output)	Previous studies	Identity function
Error function	Previous Studies	SSE
Weight optimization	Previous studies	Gradient descent with BP[11]
Generalization improvement	Previous studies	AIC and BIC[12]
Hidden layers	UAT	1
Ouput	Problem dependent	1
Hidden neurons	Trial and Error	5
Network selection criterion	Previous studies	RMSE

Table 3.9.4: Performance of the ANNs and benchmarks on three time horizons measured by RMSE x 100

Model	**S&P 500**		
Forecast horizon	1	4	8
ANN			
(Input, Hidden neurons)	(5,5)	(5,1)	(5,1)
RMSEx100	3.149	5.587	4.875
AR	3.374	5.584	4.875
RW	3.986	7.441	6.762

chosen by the criteria and the model that had the best performance measures out of sample. For example, looking at the S&P500 forecast on one year horizon, according to RMSE, AIC and BIC in-sample a (1,4) network is the best whereas out-of sample it is shown that (5,5) has the best predictive capability according to the RMSE metric. They highlight that their findings is in line with a vast amount of previous studies showing correlation between in-sample and out-of-sample is 0.2.

We also emphasise that according to Table 3.9.5 there is wide inconsistency of what architecture that was best between the different assets classes while the architectures between the different time horizons seem to have more consistency especially according to the number of input nodes.

Table 3.9.5: Architecture of the best performing ANNs for the three asset classes and three time horizons

Model	**S&P 500**			**T-Bill**			**USD/BP**		
Forecast horizon	1	4	8	1	4	8	1	4	8
(Input, Hidden neurons)	(5,5)	(5,1)	(5,1)	(2,2)	(3,5)	(3,5)	(1,5)	(2,2)	(2,1)

3.9.3 Much ado about nothing? Exchange rate forecasting: Neural networks vs. linear models using monthly and weekly data

Hann and Steurer (1996) evaluate the predictive performance of ANNs for predicting exchange rate between US-dollar and German-deutsche mark (US/DM), and benchmarks it compared to a naive and a linear regression model of the type ordinary least square (OLS). The time horizon of the prediction in this study is one month and the major focus is to evaluate the difference of predictive performance by using different time length of the exogenous inputs. The motivation is that using monthly data decreases number of data points which might lead to an overfitted network and daily data are to noisy to fundamentally predict the exchange rate. By using weekly data one could increase the data points and the nonlinear behaviour of weekly data is favoured by neural nets comparing to other models. Hence, the study comprise prediction based on data on monthly and weekly basis separately and the choice of ANN is provided in Table 3.9.6.

Table 3.9.6: Parameter choice of the optimal ANNs in Hann and Steurer (1996) research paper

Parameter type	**Method**	**Parameter choice**
Data period	Weekly/Monthly basis	1986 - 1992
Input type	Previous studies	Fundamental
# Input	SC[13] - Knowledge based	13
Preprocessing	Cointegrated time series	Cointegration model
Activation function (HL)	Previous studies	Sigmoid function
Activation function (Output)	Previous studies	Identity function
Weights optimization	Previous studies	Backpropagation
Generalization improvement	Previous studies	Cross validation & rolling window
Hidden layers	UAT	1
Ouput	Previous studies	1
Hidden neurons	PCA[14]	4

In this study the rolling window procedure was employed for both the linear and the neural net. The size of the training set was held constant and after predicting

[13]Schwarz Information Criterion

[14]Special technique for this paper of how to determine number of hidden neurons

three months ahead, the models were retrained. However, the author outline that this procedure did not lead to any significantly different results than keeping the weights constant.

Their results and evaluation were presented by comparing the results of a simpler trading strategy that utilized the outputs of these three predictions models, i.e. the estimated return. The metrics used are: Hit rate of the prediction, i.e. the probability of estimating the right direction of the return, annual return and Sharpe Ratio (SR), in this case defined as the annual return in proportion to the risk (volatility). The results using weekly (W) and monthly (M) sampling of the inputs are provided in Table 3.9.7

Table 3.9.7: Evaluation of the ANN and the benchmarks respectively models

Model	**Hit rate**		**Return**		**SR**	
Sampling frequency	W	M	W	M	W	M
ANN	55.81	64	14.75	17	6.55	3.7
Naive	44.19	59.5	-6.06	11.5	-3.25	1.8
OLS	54.65	61.9	6.90	18.7	3.58	3.5

Concluding remarks are that both the ANN and OLS outperform the Naive model according to all metrics in both cases. Further, ANN outperforms OLS when weekly data is used, although the Granger and Newbold (1977) test did not show any statistically significantly differences in prediction accuracy, whereas if monthly data is used the results are similar. The author claim that this is motivation of using ANN when there is non linearity observed in the data.

Chapter 4

Method

This chapter presents the methodology of the proposed model to answer the objective of the thesis and also how the results are to be evaluated with perspective to determine its relevance.

4.1 Data

This section describes the software used to build this model and how the data have been collected and prepared.

4.1.1 Software

The neural network model will be implemented in Matlab using the Neural Network Toolbox. The portfolio optimization will be implemented in Matlab, as well as all of the evaluation of the project. Microsoft Excel will be used as bridge to manage the data before implementation in Matlab.

4.1.2 Processing

The last data point considered in this project will be 2017-05-05. The data will be collected via the Bloomberg database, sampled at weekly frequency. Weekly data points reduces the effect of different closing times of different indices versus daily time stamps. Also, shorter time stamps tend to be more noisy, and as the goal is to estimate expected return on a quarter's horizon, daily time stamps are considered to be too short. Quarterly sampling would be dissatisfying based on the small data set such sampling would yield. The shortest times series is only available from 2003-01-03, equaling 61 data points to date at quarterly sampling. It is important to note however, even if the data is sampled at weekly frequency, returns from a longer horizon can be generated and used as input in the model. Hann and Steurer (1996) underpin the relevance of weekly sampling when predicting monthly returns. Training examples will hence be generated by sliding the input-output mapping one week ahead at a time.

For the portfolio constituents[1] the times series are available from the date as indicated in Appendix B. Some of the indices are quoted in foreign currencies. Therefore the data is converted to SEK, which is a feature supported directly in the Bloomberg software. This is done since the stakeholders of the proposed model are Swedish. As the return in an index converted to SEK is a product of the return of the index in its original currency and the return in exchange rate, the volatility will be greater in SEK. However, using the FX-converted returns is a conscious delimitation based on the great scope inflicted by the neural network estimation of the expected return.

Returns r_t will be calculated from the prices S_t from Bloomberg as log-returns in order to keep additivity in time, i.e.

$$r_t = \ln\left(\frac{S_t}{S_{t-1}}\right). \tag{4.1.1}$$

The approximation that a quarter is equal to 12 weeks (a month is approximated with 4 weeks) will be made to simplify data management, i.e. the return for a quarter is approximated with

$$r_t^{(q)} = \sum_{i=0}^{11} r_{t-i}. \tag{4.1.2}$$

The time period will be partitioned into two major data sets. The first data set will be allocated to model selection purposes, i.e. functional approximation of neural network input/output mapping for expected value estimation and selection of stochastic process for asset returns. Consequently, the first data set, model selection set, will consist of training and validation set in regards to neural network terminology. The second data set will be used to do the final testing of the neural network predictive performance, as well as by the portfolio optimization model. As such, the second data set corresponds to the test set of a neural network, as well as the period where portfolio weights are optimized. The second data set is also the set where the total model will evaluate whether it could generate excess return on the market or not.

The portfolio optimization will be run approximately for 4 years, and set to 16 quarters, i.e. 192 data points. As such, the test set will be partitioned to consist of data points after 2013-08-30. This partitioning is decided with motivation to create enough data points to train the network while it leaves enough data points to evaluate it. For the model selection purposes, the length will vary per asset according to the available data.

4.2 Phase 1: Neural network

A phase 1 decision tree is presented in Figure 4.2.1 containing decisions that will have to be made when creating the neural network.

[1]Note that the terminology portfolio constituents, assets and indices refer to the asset class indices whose expected return is to be estimated by the neural networks as well as managed by the portfolio optimizer.

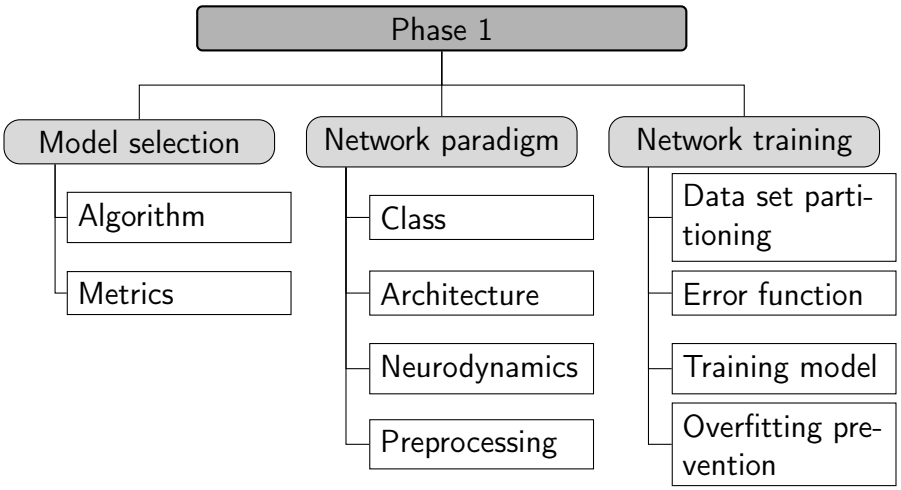

Figure 4.2.1: Overview of the decisions included in phase 1

4.2.1 Model selection algorithm

Determining values of the great number of hyper-parameters will be done by either manual search or the model selection algorithm (MSA). Manual search will be applied to the hyper-parameters with a choice that has consensus in the theory and the rest will be left to the MSA. To determine the set of hyper-parameters that the MSA is optimizing over, random search will be applied. The rationale for using random search is that the random search has been shown by Glorot and Bengio (2010) to find better hyper-parameters in the same amount of trial points than the grid search. The size of the set of hyper-parameters that MSA will evaluate, from the random search picking, will in this report be named as the number of random searches [2].

The random search will feed the MSA with an amount of hyper-parameters within a prespecified relevant interval where its limits will be decided through extensive tests in the model selection data set. The MSA will evaluate networks with different combinations of hyper-parameters. Consequently, increasing the total number of random searches, S, will increase the size of the optimization and the total computational time. On that basis the hyper-parameters will be delimited in such way that the networks will be computed in a reasonable time ($\approx$ one night, 8 hours, optimization).

Hence, the number of random searches, and subsequently the hyper-parameters set, for each hyper-parameter will be determined in such way that its large enough to evaluate values that might be optimal but still does not exceed a relevant number of random searches in regards to computational time.

Different choices of hyper-parameters will be done for every portfolio constituent according to a model selection algorithm for each of the constituents.

The random search will draw uniformly distributed random variates in order to generate S random searches $\boldsymbol{\lambda}^{(1)}, \boldsymbol{\lambda}^{(2)}, \ldots, \boldsymbol{\lambda}^{(S)}$ for the neural network model. The optimal choice of hyper-parameters $\boldsymbol{\lambda}^*$ will be selected according to

[2]Example: A random hyper-parameter, $\theta \in \{1, 2, 3, 4, 5\}$, with random search picking and two random searches, will feed the MSA with two alternative choices

$$\boldsymbol{\lambda}^* = \underset{\boldsymbol{\lambda} \in \{\boldsymbol{\lambda}^{(1)} \dots \boldsymbol{\lambda}^{(S)}\}}{\arg\min} \; \Psi(\boldsymbol{\lambda}) \tag{4.2.1}$$

where the generalization error $\Psi(\cdot)$ will be determined as a sum of the validation set errors, $J_{validation}(\mathbf{W}_{k|\boldsymbol{\lambda}})$, in the k:th cross-validation sets according to

$$\Psi(\boldsymbol{\lambda}) = \sum_{k=1}^{M} J_{validation}(\mathbf{W}_{k|\boldsymbol{\lambda}}) \tag{4.2.2}$$

where the weights $\mathbf{W}_{k|\boldsymbol{\lambda}}$ in cross validation set $k \in \{1, 2, \}$ is conditional on $\boldsymbol{\lambda}$. The validation set error function is defined in Section 4.2.7. There are K-folds of the cross-validation. The choice of using cross-validation as method for evaluating the hyper-parameters is justified by the decreased randomness in generalization error by using several validation sets instead of one. The number of folds, K, will be chosen to 5. As such, for each cross-validation run, the neural network will be trained on 80 % and validated on 20 % of the model selection data. This ratio between training and validation is approximately the same as with the common partitioning of 70-15-15 % in training-validation-test set, (i.e. $\frac{80}{20} = 4$, $\frac{70}{15} \approx 4, 7$).

As the $\boldsymbol{\lambda}^*$ will be a source to 5 different set of weights $\mathbf{W}_{k|\boldsymbol{\lambda}^*}$, one for each cross-validation fold, the optimal set of weights will be decided according to

$$\mathbf{W}^*_{\boldsymbol{\lambda}^*} = \underset{\mathbf{W}_{m|\boldsymbol{\lambda}^*}}{\arg\min} \; J_{validation}(\mathbf{W}_{m|\boldsymbol{\lambda}^*}), \quad m = 1, \dots, M. \tag{4.2.3}$$

Deciding which set to use could be done by the ensemble averaging methodology presented by Enke and Thawornwong (2005) but is considered inappropriate and thus neglected, since it is a try to create a good solution by averaging a couple of bad solutions. Instead, we choose the one which is best according to its validation set.

Data partitioning

The data set will be randomly permuted in order to deal with the risk of biasing the neural networks towards certain market conditions. The random permutation will be held constant for all testing of hyper-parameters, and training. Once the data set is randomly permuted it will be divided into five equally sized blocks, where each of the blocks will act as the validation set in the five cross-validation runs, with the remaining four blocks as the training set.

4.2.2 Class

A feedforward network will be used as the class for this study which align with the most successful previous studies. Feedforward network is according to Haykin (2009) more stable then alternatives such as a recurrent network. Further, this study has the learning task of a hybrid between function approximation and prediction for which Haykin (2009); Hagan et al. (2014) recommend the multilayer feed forward class whereas a

recurrent network is more applicable in a controlling problem. A feedforward network has another benefit of enabling the use of the successful back propagation training model which also motivate this choice.

$$\hat{y}(t) = f(t) = \varphi_o\left(\sum_{j_{L-1}=0}^{M_{L-1}} w_{j_{L-1},j_L}^{(L-1)}\varphi_h\left(\cdots\sum_{j_2=0}^{M_2} w_{j_2,j_3}^{(2)}\varphi_h\left(\sum_{j_1=0}^{M_1} w_{j_1,j_2}^{(1)}x_{j_1}(t)\right)\right)\right) \quad (4.2.4)$$

4.2.3 Architecture

Most of the hyper-parameters of the architecture will be defined by the model selection algorithm since it is easy to vary the parameters and difficult to upfront set a specific value. Intervals will be determined through tests and with endorsement from studies within optimization the hyper-parameters will be specified with random search and subsequently determined through performance evaluation, i.e. MSA. Some hyper-parameters with a choice that is consensus in the literature will be pre-defined in order to decrease the total number of random searches and thus the computational time, to increase focus on optimizing the most crucial hyper-parameters.

Input

Our methodology of determining the input vector, $\mathbf{x}$, will be divided into three parts. First, a bank of potential information holders will be collected. Secondly, a relevance analysis will be conducted to determine which of these inputs that contain information about each portfolio constituent's future return. The relevance analysis will be done by a two-sided Pearson correlation test between the collected bank of potential inputs and the index whose return to be estimated. Pearson correlation will capture linear and monotonic[3] relationships. It will though not capture non-linear, non-monotonic relationships, but in absence of good metrics for such relationships a delimitation in the relevance analysis is made to use the Pearson correlation metric. The Pearson correlation test is used to test the null hypothesis $H_0 : \rho = 0$ vs. the alternative hypothesis $H_1 : \rho \neq 0$. This test holds if the underlying variable has a normal distribution, whether that is true or not is not considered to be crucial regarding the purpose of the test. However, under the null hypothesis it holds that the following test statistic has a Student's t-distribution

$$T = \rho\sqrt{\frac{M_1 - 2}{1 - \rho^2}} \sim t(M_1 - 2). \quad (4.2.5)$$

(Rahman, 1968)

A correlation of 0.088 given 500 points of estimation will violate the critical limit of the test and hence the null hypothesis will be rejected, and the input will be concluded to contain useful information. Finally, to reduce the number of inputs used but keep

[3]The Pearson correlation will not yield a metric of 1 of a perfect monotonic relationship, but will still be separated from 0, which is the test that is conducted.

as much information as possible a PCA will be conducted where the number of eigenvectors to be used will be determined by the random search algorithm. The limits for the random search will be determined by conducting tests in the model selection data set for different architectures and assets.

The inputs collected to our bank will consist of fundamental and technical data. As fundamentals, macroeconomics will be used to capture global economic activities which most likely have an impact of our portfolio constituents. The macroeconomic factors that will be used has been compiled together with Söderberg & Partners and the team working with macro economical analysis, see Appendix B.1, and will be collected via the Bloomberg database.

Since our portfolio constituents are macroeconomic indices themselves, these will also be a part of the bank of potential inputs. As such, the portfolio constituents will undergo the relevance analysis, to determine whether they have impact on index to be estimated or not. The input with shortest available data points will have direct impact on the limit of the training set. Since some of the times series are considerably shorter than others, the input bank will only consist of those indices that are at least as long as the times series for the return to be estimated. So, the bank of potential inputs will be specific to each of the neural networks. This is to be enable to keep as many training examples as possible.

Both microeconomics and business specific variables are irrelevant since indices are used as portfolio constituents. Consequently, factors from the multi-factor models are neglected.

The technical inputs will consist of time lagged returns of the specific index return, $r^{(Q)}_{t+1Q}$, to be estimated. This will according to Cont (2001) capture the existing autocorrelation that exists between present and historical time periods. Due to the fact that we are estimating on a quarterly basis and uses data with weekly intervals the returns used in the tapped delay line and eventually the neural network will be converted to monthly and quarterly returns. Weekly historic returns might lack information of the return during the coming quarter, even though we are lagging several weeks back, with consideration of the importance of limiting the number of inputs. As such, the conclusion is drawn that lagging months and quarters will render more information per lagged input used. Quarterly data will be used as previous periods return and monthly data will be used to capture potential technical momentum of the index. Lagged data will reduce the training data proportional to the lagging period, conclusively, a tapped delay line with lagged returns up to 36 weeks (0-2 quarterly returns lagged, 0-8 monthly returns lagged) before the specific date to be estimated will be used and fed to the bank of potential inputs. Figure 4.2.2 show the tapped delay line for monthly returns (M), and equivalently will be done for quarterly returns. Remember the fact that $r^{(Q)}_{t+1Q}$ is to be estimated a quarter in forward which implies that the first return $r^{(Q)}_{t}$, i.e, lagg 0, used as input is 12 weeks before the targeted return.

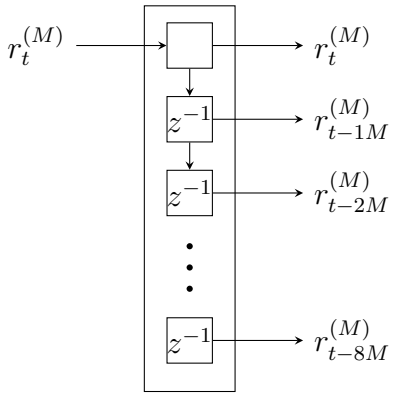

Figure 4.2.2: Visualization of the tapped delay line for monthly returns, where $r_t^{(M)}$ is the latest month's return, and $r_{t-1}^{(M)}, \ldots, r_{t-8}^{(M)}$ are the time lagged monthly returns.

The specific lags to be used in the network will be determined through the relevance analysis. Returns from the other portfolio constituents will also be used as technical inputs without using a tapped delay line.

If, according to the theory, the currently best models features predictive capabilities when it comes to estimate the expected return this is useful information we wish to use. Consequently, the networks for all portfolio constituents except the FX-rates will be fed with the next period estimation from CAPM. CAPM is not suitable to estimate FX-returns hence, the networks to predict the expected return for FX-rates will be fed with 0 from the random-walk model. Any spectacular successful results regarding their predictive capability have not been found though, but they have not been proven beaten with any other model and are the benchmarks for our networks. However, if they can provide information the networks will extract it.

All other technical indicators are neglected with motivation that these indicators are just processed returns and will not provide additional information than the lagged returns.

In Table 4.2.1 there is a summary of the bank of potential inputs. Keep in mind that this is the maximal number of inputs fed to the bank, constituents that has more available data points will have a reduced number of inputs, as discussed previously. The bank of potential inputs can be divided into three subgroups. The technical indicators refers to the subgroup consisting of lagged returns of the index which return is estimated. Index returns is the subgroup of the portfolio constituents where the last quoted quarterly returns are being used. The macro factors consist of the latest quoted macro economical time series. Lag number refer to the time lag from the estimation date.

Table 4.2.1: Summary table of the bank of potential inputs for the neural networks respectively.

Subgroup	Time period	Lag	#	Unit
Technical Indicators				
Lagged returns	Monthly	0-8	9	Log-return
	Quarterly	0-2	3	
CAPM/RW[4]	Quarterly	0	1	Log-return
Index returns	Quarterly	0	14	Log-return
Macro factors	Weekly/Monthly/Quarterly[5]	0	39	Index Point/Relative change[6]

Hidden layer

The number of hidden layers will set the number of total layers, L, since the output layer and input layer is pre specified to be one respectively, hence the number of hidden layersL_h will be $L-2$. This parameter will be determined through the model selection algorithm with pre-specified limits from tests. A single-layer feed forward neural network, i.e. no hidden layer, does not have the capability to capture non-linearities which exclude the possibility of using a SLFFNN. Further, the UAT says that one hidden layer is enough which also is the most frequently used of previous researchers. Chester (1990) though claims that there could be beneficial to increase the hidden layers to decrease the number of hidden neurons in the first hidden layer drastically. There is also incentives to enable the model selection algorithm to try one more hidden layer from the theorem stated by Kolmogorov. The test will also evaluate whether a third hidden layer might be of interest in favor of reducing neurons in the prior two hidden layers. Hence, the number of layers will be $L \in [3,4]$ or $[3,5]$, i.e. number of hidden layers will be $[1,2]$ or $[1,3]$.

The number of random searches from the different possible amount of hidden layers will be proportional to the proportion of possible neuron combinations for that number of hidden layers and the total number of possible combinations for all hidden layers choices[7]. This with motivation to avoid that the random search only feed the model selection algorithm with architectures with the same amount of layers.

Hidden neurons

There is no consensus regarding the optimal number of hidden neurons, and the number of neurons will consequently be determined by the model selection algorithm choosing

[4]Will be an input regardless of the results in Pearson correlation test

[5]See Appendix B.1 for the frequency of each macro factor.

[6]Depends on the nature of the metric. See Appendix B.1 for the unit of each macro factor.

[7]Example: [1-2] hidden layers tested with 1-20 neurons in the first layer, 1-5 neurons in the second layer. Possible combinations for one layer is 20, possible number of combinations for two layers = 5*20 = 100. One layer will be tested $\frac{20}{120}$, two layers will be tested $\frac{100}{120}$, of the total number of random searches.

from number of hidden neurons from the random search method. The limits for the random search algorithm will be determined through tests in the model selection data set. According to the theory increasing the architecture with one layer should reduce the number of neurons in the previous layer. Thus, depending on the results, the limits of neurons for each layer will be set with conditions that depends on the number of hidden layers.[8] This also reduces the high number of possible combinations with three layers that otherwise would increase the computational time for the MSA.[9]

Output

In order to minimize the total error there will be one architecture, estimating the expected return, for each portfolio constituent. This will make the most accurate prediction instead of the alternative creating one network with multiple outputs. Further, since the learning task is a regression problem and more specifically a function approximation/predicting problem there is no need of multiple outputs describing specific classes or clusters.

4.2.4 Preprocessing

The way the data is preprocessed is dependent of the choice of activation function and specifically the range of what value the output of the activation function can adopt. This holds especially in the case where a multilayer class of network is used. Remember, the output of the general node in the arbitrary layer will subsequently become the input to the next layer, therefore their limits should conform. However, in order to not bias the gradients to any specific direction according to LeCun et al. (1998), a mean of the inputs around zero is wanted. Thus, the range of normalization is set to $[-1, 1]$. Every input times series x_i will be processed according to

$$x' = 2\frac{x - min_x}{max_x - min_x} - 1. \tag{4.2.6}$$

This preprocessing procedure will be conducted both before the PCA to ensure that the PCA is not skewed towards a variable with a unit that gives higher values than the other, as well as after determined the principal components to ensure that the information that is fed into the network is in $[-1, 1]$. The target data of the times series will not be normalized to $[-1, 1]$ and instead handled with an unlimited activation function (identity function) in the output node.

One assumption of CAPM that aligns with our perspective of the capital market is that an investor should get compensated by the exposure of risk of a specific asset, r_a, commonly mentioned as a risk premium, r_p. This premium hence correspond to the excess return over the riskfree rate, r_f, defined as

[8]Example: the limit of neurons in the first layer could be set to 10 with condition that the architecture consists of one hidden layer and 5 otherwise.

[9]Example: architectures with size up to 3-2-1 gives 6 potential 3-hidden layer-combinations, instead reducing first layer and second layer to 2 and 1 respectively if a third layer is utilized gives 2 different combinations

$$r_p = r_a - r_f \tag{4.2.7}$$

where r_f will be set to STIBOR3M. Consequently, this assumption implies that the return is dependent of the riskfree return, which during the latest years has decreased tremendously. To capture these levels the risk premium of each portfolio constituent, $r_{p,i}$ will be used as inputs and desired outputs of the network instead of the actual return. The returns that will be used as inputs will be pre-processed with the riskfree rate before they are normalized an fed into the PCA. Enke and Thawornwong (2005) did a similar thing, as they predicted the return of S&P500 less the riskfree rate. The FX-rates will not be preprocessed according to (4.2.7) as CAPM is not valid for FX-rates. The length of the STIBOR3M data series will consequently limit the possible number of training examples from 1987-01-02 for the indices, but not for the FX-rates.

4.2.5 Postprocessing

As the desired outputs of the neural networks are processed by subtracting the riskfree rate, the output of the neural network given an input vector has to be postprocessed in order to get the expected return for the asset. As such, the estimated expected return $\hat{\mu}$ for the asset is given by

$$\hat{\mu} = \hat{y} + r_f \tag{4.2.8}$$

where $\hat{y}$ is the output from the neural network and r_f is the riskfree rate. Similarly, this postprocessing will not be conducted for the FX-rates, since they are not preprocessed with the riskfree rate.

4.2.6 Neurodynamics

One of the purposes of using ANN instead of other machine learning models is the capability of capturing nonlinearities in the data. Further, the training, i.e. the optimization to find the best model demands differentiability. Therefore the neurodynamics in the hidden units will consist of a sigmoid type of activity function. Since the output of the hidden nodes is the input to the next layer there is of interest to keep the output in the range of $[-1, 1]$, which according to the literature also decreases the computational complexity of the network training. Hence, the activation function in the hidden neurons from the general input, z, will be the TanH-function

$$\varphi_h(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}. \tag{4.2.9}$$

As the nonlinearity is captured through the hidden layers there is no requirement of using a non-linear activation function in the output node according to the studies of Qi and Zhang (2001); Hann and Steurer (1996). Further, the Universal Approximation Theorem also holds for a linear activation in the output node. From Section 4.2.4 there was motivation to use the identity function as output activation function. With further consideration that the learning task of the problem is regression and the network should

output an expected return of any level, the identity function without any saturation as

$$\varphi_o(z) = z \tag{4.2.10}$$

is motivated and will be used. The functions φ_h and φ_o are differentiable, which is a requirement for using the backpropagation algorithm of the derivative of the error function with regards to the weights.

4.2.7 Error function

Commonly in the field of machine learning two sources of errors are discussed, bias and variance, described in Section 3.2.1. Ideally, one wish to reduce the bias while at the same time decreasing the variance. A reduced bias, i.e. reduced systematic error in the model, is of interest to avoid an underfitted network that has limited predictive capabilities out of sample. In contrast, a reduced variance is desirable since higher variance imply potential to represent the training data but to the expense of increased risk of overfitting the data. Further, a higher variance of the estimated return will make the optimization model reallocate the assets to a bigger magnitude. Unfortunately reducing both simultaneously is impossible, and is also not the prime objective. There is a trade-off between finding a good estimate, and the variance of ditto. In this context these errors will prohibit the networks capability to generalize beyond the in-sample data set. Due to the fact that the model will be optimized over both the training and validation set, i.e. weights optimization and hyper-parameter optimization, the same error function will be used for both the training and validation set. This means that the optimization will step in the descent direction in the weight space in regards to the error functions for both the training and validation set before it starts to overfit, which is prevented by the early stopping technique, see Section 4.2.9.

We will perform extensive testing of the neural network performance in the model selection data set in order to determine an appropriate error function. The most common error function in the previous research of estimating the expected return is the mean or sum of squared error (MSE/SSE)[10]. This will naturally serve as a starting point of testing neural networks. However, in order to investigate how the dynamic of the estimations changes in-sample when the exponent b of the error changes, we will test different error exponents, and see how the dynamic (variance, bias) of the estimations changes, see (4.2.11).

$$J_{training}(\mathbf{W}) = \sum_{m_t=1}^{N_t} (|y_{m_t} - \hat{y}_{m_t}|)^b, \quad b > 0 \tag{4.2.11}$$

Exponents lower as well as higher than the common SSE will be tested according to $b \in \{0.1, 0.5, 1, 2, 6, 26\}$. If any of the "extreme" exponents show promising characteristics, further exponents might be tested.

[10]The MSE and the SSE will give the same solution as the objective function is only scaled by a constant

One property of CAPM that is desired is its low variance, therefore a test to steer the optimization towards CAPM to lower the hypothetical high variance of the estimations will be conducted. This is also an action to prevent potential overfit of the model selection set which is further discussed in section 4.2.9. As the neural network estimates the return less the riskfree rate the model has to be to steered towards CAPM less the riskfree rate.

$$J_{training}(\mathbf{W}) = \sum_{m_t=1}^{N_t} (|y_{m_t} - \hat{y}_{m_t}|)^b + \lambda_{CAPM} \sum_{m_t=1}^{N_t} (CAPM_{m_t} - r_{f,m_t} - \hat{y}_{m_t})^2 \quad (4.2.12)$$

Analogous with the reason of not using CAPM as inputs to the FX-networks, described in 4.2.3, penalizing the deviation from CAPM for these estimations when training and selecting network would be inappropriate. Consequently, the error function in 4.2.12 is modified as

$$J_{training}(\mathbf{W}) = \sum_{m_t=1}^{N_t} (|y_{m_t} - \hat{y}_{m_t}|)^b + \lambda_{RW} \sum_{m_t=1}^{N_t} (\hat{y}_{m_t})^2. \quad (4.2.13)$$

The coeffiecients, λ_{CAPM} and λ_{RW}, i.e. the proportion of how much the error function will penalize deviation from CAPM and FX respectively, will be tested with the values $\lambda_{CAPM}, \lambda_{RW} \in \{0, 0.5, 1, 1.5, 24\}$. The objective is to find the coefficient that yields an appropriate volatility in the model selection data set. The coefficients might be set individually for each index depending on the dynamic of the estimations in the model selection set.

The objective of these tests aligns with the common dilemma of bias and variance discussed above, define an error function for each index where the estimations in model selection set has a volatility less than 15% than the volatility of the actual returns, but still has as low bias error as possible. This limit of volatility means that the estimations given a 95 % confidence interval will roughly[11] vary between the mean and $\pm 2\sigma$. This level is considered as appropriate for the variation of the expected value estimations, as well an appropriate level to limit the re-allocations of the portfolio optimization.

4.2.8 Training

Although the theory presented in Section 3.6 point out that online learning is better than batch learning, the batch learning implementation by Matlab have been shown (Demuth and Beale, 2002) to usually be more efficient[12]. The two arguments in the theory is that online learning handle data redundancies efficiently, and do potentially avoid local minimas. The first argument seems to be overcome by the Matlab implementation, ans the second argument is handled in our model by having several initial conditions. As such, we will use batch learning gradient descent with adaptive learning

[11] Given a normal distribution, samples will vary between the mean and $\pm 2\sigma$ given a 95 % confidence interval

[12] The actual reason why is not documented by Matlab, and not further investigated by the authors

rate. As the learning rate will be adaptive, the initial choice of ditto is not crucial. As such, it will be set to the standard in the Matlab Neural Network Toolbox, which is 0.01 (Demuth and Beale, 2002). The learning rate will be adaptive to the changes of the error function. If the error function decreases, the learning rate will be increased by a factor of 1.05 in order to keep speed up learning as the magnitude of the gradient decreases. If the error function increases after one epoch, which means that the weight step was too large, the weight update is discarded and the learning rate is decreased with the factor 0.7. This choice stems from lack of consensus in the literature regarding this parameter and the fact that these are the standard choices in the Matlab Neural Network Toolbox (Demuth and Beale, 2002).

The weight optimization will be interrupted if any of the following conditions are disrupted:

- The overfitting criterion is met (see Section 4.2.9)
- The length of the gradient is smaller than 10^{-35}
- The number of epochs completed is larger than 1000 (Prechelt, 1998)

The weights will be initialized using the model of Nguyen and Widrow (1990) to initialize neuron and bias weights, described in Section 3.6.6. The main rationale for using this initialization model is that few neurons are wasted, since all the neurons are in the input space, and that training works fast, since each area of the input space has neurons. (Demuth and Beale, 2002)

Number of initial conditions

As the optimization problem for estimating the weights of the neural networks is non-convex we will initialize the weights several times to enable converging to different local minimums of the weight space. Optimally the number of inital conditions would be very large to enhance the possibility of reaching global optima but with restriction of computational time every set of hyper-parameters will be run 9 times. This is within the interval suggested by Bengio (2012) (original suggestion 5-10 times re-initialized initial conditions). The weight set with the lowest validation error for the 9 runs with different initial conditions will be chosen. The ensemble averaging methodology is considered inappropriate, since it is a try to create good solutions by averaging a couple of bad solutions. Instead, we choose the one which is best according to the validation set.

4.2.9 Overfitting prevention

On top of potentially regularizing the error function with the CAPM-error, early stopping will be conducted to decrease the risk of overfitting. Early studies, including Enke and Thawornwong (2005) have utilized the early stopping technique for preventing overfitting.

By conducting an initial test, we conclude that the error on training and validation set relative to the number of epochs for this problem setting seem to follow the theoretical appearance, i.e. when the validation error starts to increase it will not decrease again.

Hence, our starting point is to stop the training as soon as the validation error starts to increase. However, to make sure that the validation set error does not start to decrease again, the algorithm will be run for 6 more epochs after the validation error has started to increase to make sure that it does not decrease again, which is in line with Matlab Neural Network Toolbox recommendation. As such, the techniques proposed by Prechelt (1998) will not be utilized.

4.3 Phase 2: Portfolio optimization

This section will describe the chosen portfolio optimization model, including external preferences set by S&P. The portfolio optimization will be run on data starting from 2013-08-30.

4.3.1 Foreign exchange

As the returns for the indices used in this thesis will be in SEK, and include the impact of FX-rates changes, the portfolio will be exposed to FX-risk. As such, it will not make any sense to keep FX-rates as possible investments as these would only increase the FX exposure. Also, additional assets would have to be included in the portfolio as investments of the invested foreign currency. As such the portfolio constituents will be reduced from the 15 different neural networks in phase 1, to 12 different portfolio constituents in phase 2. The FX-rate estimations will only be evaluated as a part of the first phase.

4.3.2 External specifications

The following bullets summarizes the properties that the asset allocation generated by the portfolio optimization model should comply with:

- Swedish investor - i.e. the return in SEK is of interest
- No shorting - i.e. no negative holdings of assets
- No borrowing - i.e. no negative holding of cash
- Differentiated portfolio - i.e. holding a broad range of the portfolio constituents at every time step
- Moderate re-allocations - portfolio turnover[13] of maximum 0.10 at each rebalancing and 0.40 per annum

4.3.3 Portfolio optimization model

The portfolio optimization model will be a two-stage stochastic programming model without recourse decisions based on the sum of the utility of the wealth in a set of S scenarios. The advantage versus the widely used mean variance model is that the utility function of the investor can be set to something else than the quadratic utility

[13]Sum of the absolute buy- and sell decisions divided by 2. If the entire portfolio is re-allocated, the portfolio turnover is 1

function, as well as, handle market imperfections such as transaction costs. Also, the stochastic programming model allows for other distribution assumptions of the asset returns than the normal distribution. The mean variance model is sensitive to the fluctuations of the estimated expected return, and tends to hold only a few of the portfolio constituents (Black and Litterman, 1992). The portfolio optimization will be run once every quarter and will consequently cause re-allocations quarterly[14], in accordance with S&P advising intervals.

The parameters of the portfolio optimization problem is summarized as follows:

Parameters	
I	Number of assets
S	Number of Monte Carlo simulated scenarios
$i \in \{1, \ldots, I\}$	Number i of the asset x_i in the portfolio universe
$s \in \{1, \ldots, S\}$	Number s of the Monte Carlo generated scenarios
Δt	Monte Carlo simulation time period
x_i^{init}	Initial holding in asset i
$r_i^{(s)}$	Return for asset i in scenario s in SEK
c^{init}	Initial cash
r_f	Return for the riskfree asset in SEK
p_s	Probability for scenario s occuring
γ	Risk aversion parameter
τ	Transaction cost

The variables of the optimization problem is buy and sell decisions for each asset which will be denoted:

Variables	
x_i^{buy}	Buy decision for asset i
x_i^{sell}	Sell decision for asset i

The objective function of the portfolio optimization will be based on the sample average approximation of the wealth in a set of scenarios. The scenarios will be generated by Monte Carlo simulating asset returns using a copula function. The wealth in each scenario s will be the sum of the value of each asset i, $x_i^{holding} e^{r_i^{(s)}}$ plus the capitalized cash $ce^{r_f \Delta t}$ (equivalently the riskfree asset).

[14]Will be approximated with 12 weeks as the data is sampled on weekly frequency

Objective function

$$\begin{aligned} \max z &= \max \sum_{s=1}^{S} p_s U(\sum_{i=1}^{I} x_i^{holding} e^{r_i^{(s)}} + c e^{r_f \Delta t}) \\ U(W) &= \begin{cases} \ln(W) & \text{for } \gamma = 0 \\ \frac{W^\gamma}{\gamma} & \text{for } \gamma \leq 1, \gamma \neq 0 \end{cases}. \end{aligned} \tag{4.3.1}$$

A set of constraints will be formulated, and is presented below:

Constraints

New holding:
$$x_i^{holding} = x_i^{init} + x_i^{buy} - x_i^{sell}, \quad i = 1, \ldots, I \tag{4.3.2}$$

No arbitrage:
$$x_i^{buy}, x_i^{sell} \geq 0, \quad i = 1, \ldots, I \tag{4.3.3}$$

No shorting:
$$x_i^{holding} \geq 0, \quad i = 1, \ldots, I \tag{4.3.4}$$

New cash:
$$c = c^{init} + \sum_{i=1}^{I} x_i^{sell} - \sum_{i=1}^{I} x_i^{buy} - \tau \sum_{i=1}^{I} (x_i^{buy} + x_i^{sell}) \tag{4.3.5}$$

No borrowing:
$$c \geq 0 \tag{4.3.6}$$

When the portfolio optimization is run, the relative weights in each holding is computed as

$$w_i = \frac{x_i^{holding}}{\sum_{k=1}^{I} x_k^{holding} + c} \tag{4.3.7}$$

and the relative weight in the riskfree asset is determined by

$$w_c = \frac{c}{\sum_{k=1}^{I} x_k^{holding} + c}. \tag{4.3.8}$$

4.3.4 Scenario generation

The scenarios will be generated using a copula function and univariate distributions for each asset. The assumption will be made that the correlation is constant, and

consequently the copula correlation P will be estimated using maximum likelihood estimation in the model selection set. The copula function, out of gaussian and student's t, that yields the highest log-likelihood value will be chosen to estimate the copula function.

As financial times series do show heavy tails, but have an aggregational gaussanity on longer time periods, it is not clear which of the processes in Appendix D that suits the data best. As the simulation period will be a quarter, the data will most probably show aggregational gaussanity, but to what degree is unclear. Also, because of the fact that volatility clustering exists, we will use the GARCH(1,1) model for estimating the volatility. Therefore, the Geometric Brownian Motion, a student's t process and the GARCH-Poisson process will be tested on historic data in the model selection data set. Their suitability to describe the distribution of the data will be compared through QQ-plots (Quantile-Quantile plots). In the QQ-plot, the quantiles of a sample that is transformed via the inversion principle is scattered against the theoretical standard normal quantiles. If the scatter pattern approximately forms a straight line, the distribution assumption is approximately correct. The QQ-plots will be used to evaluate several distributions by visually inspecting the QQ-plots, and the distribution that generates the most linear relationship will be chosen. If several QQ-plots are similar, the process with the fewest parameters is chosen.

For the GBM, the expected value of the return μ_t will be estimated using the neural networks, and the volatility σ_t will be estimated using GARCH(1,1), where the parameters in GARCH(1,1) will be estimated using MLE. These methods will be used likewise to determine the student's t process, its degrees of freedom, v, though will be estimated using MLE. For the GARCH-Poisson process, μ_t will likeweise be the output of the neural networks, the rest of the parameters will be estimated through MLE. See Appendix D for detailed explanations of the respective univariate distributions.

Once the most appropriate univariate distributions are determined, the scenarios will be generated using a single step-ahead Monte Carlo simulation with the time step Δt corresponding to a quarter, i.e. $\Delta t = \frac{12}{52}$ based on our approximation of a quarter.

By assuming that the distributions used to generate scenarios holds approximately, the probability for each scenario to occur is the same, i.e. the probability $p_s = \frac{1}{S}$, see Section 3.7.3. The number of scenarios to be generated will be determined based on the computational time, as it is a trade-off between a more complete description of the possible outcomes and the computational time. The more scenarios that are generated, the less risk will be invoked by the portfolio optimization. But since a more complete description of possible outcomes is desired, we will choose the number of scenarios S to be as many as possible while still keeping a reasonable computational time. This is based on the decreasing marginal utility of wealth when $\gamma < 1$, i.e.

$$\frac{\partial U(W)}{\partial W} = \gamma \frac{W^{\gamma-1}}{\gamma} = W^{\gamma-1} > 0, \quad \gamma < 1, \gamma \neq 0 \tag{4.3.9}$$

$$\frac{\partial^2 U(W)}{\partial W^2} = (\gamma - 1)W^{\gamma-1} < 0, \quad \gamma < 1, \gamma \neq 0 \tag{4.3.10}$$

and the same holds for $\gamma = 0$.

In order to be able to reduce the variance of the generated scenarios, the variance reduction technique Latin Hypercube sampling will be used (Shapiro et al., 2009). This technique divides the $[0, 1]$ in S equally sized intervals, and draws one uniformly distributed random variate at each interval. These random variates will then be used to create random variates from any distribution using the inversion principle. As such, Latin Hypercube guarantees to draw random variates more evenly spread out, in every interval, over the probability space.

4.3.5 Estimating the portfolio optimization parameters

In order to optimize the portfolio, a couple of parameters that are not already explained have to be estimated, namely r_f, τ and γ.

As the optimization faces a Swedish investor, the riskfree rate $r_{f,t}$ at time step t will be estimated with STIBOR3M. As STIBOR3M is quoted as a simple annualized rate with 3 months maturity, the rate is converted to a continuous rate according to

$$r_{f,t} = \ln\left((1 + \frac{1}{4} r_{STIBOR3M,t})^4\right) \tag{4.3.11}$$

which holds approximately disregarding the fact that the number of days differs between months. Differences in the number of days in a month will have a very small impact and thus neglected.

We will assume that the trading is done in such way that the model buy at sell rates and sell at buy rates at the stock close when the portfolio optimization is run. As such, the transaction cost, τ, can be determined by calculating the historic spread between buy and sell rates. According to S&P, fund trading often costs 500-1 000 SEK per order. However, the fix cost will be neglected, since the portfolio will be optimized using a relative wealth. S&P also claim that ETF and equity trading cost 3-5 bps[15] of order volume. As the fix costs were neglected, the upper limit of the interval, i.e. 0.0005, is assumed to approximate the transaction costs for all the assets in the optimization phase.

The risk aversion parameter γ will determine the utility function $U(W)$ and will control how much risk that will be taken by the portfolio optimization. In practice, every investor have a different $U(W)$, and S&P determines this by conducting a survey where the customer have to take stand to a number of financial options built on the methodology in Hanna et al. (2001). Based on the answers, the customer is assigned one of seven risk levels, where the customer has a power-utility function with $\gamma \in \{-0.50, -1.46, -2.99, -5.29, -7.70, -11.20, -27\}$. Consequently, we will run seven different portfolios in phase 2, one for every risk level.

[15] 1 bps = 0.0001

4.3.6 Solving the portfolio optimization problem

As previously shown in (4.3.9) and (4.3.10), the utility function $U(W)$ has a strictly positive first derivative and a strictly negative second derivative which means that the function is concave. As the objective function (4.3.1) is a sum of utility functions $U(W)$ multiplied with a constant p_s, the objective function is also concave. All of the constraints to the optimization problem are linear, which means that they form a convex set. In conclusion, this means that the optimization problem is convex (Lundgren et al., 2008), and as such a local maximum of the deterministic equivalent to the stochastic programming problem is the global maximum of the problem.

As the objective function is non-linear, an appropriate method has to be used to solve the optimization problem. A primal-dual inner point solver will be used in Matlab, which is explained by Alizadeh et al. (1998).

4.4 Evaluation

The evaluation of the model will determine whether the objective of this thesis is fulfilled or not, which underpin its importance. Both phases will be evaluated on the test set simultaneously after both phases are done. This is important to not enable any modification of the portfolio optimization model according to the evaluated characteristics of the estimated return. As such, the evaluation will indicate the true out-of-sample performance.

4.4.1 Phase 1

The evaluation of the first phase will consist of three steps: First, we will create benchmarks, as it is of interest to evaluate whether our model is better than other existing models. Secondly, performance metrics are calculated. The last step is to conduct a statistical test of the significance levels of possible improved estimation accuracy in our model compared to the benchmarks. This phase will be evaluated using data on a weekly frequency to get a decent number of test data points.

The benchmarks of this phase have been set with two objectives. First, evaluate our model with comparison to the performance in the literature. Hence the first benchmark that will be used is linear regression which is defined as

$$\hat{\mu}_{b1}^{(q)} = \alpha_0 + \alpha_1 x_{1i} + \alpha_2 x_{2i} + \ldots + \alpha_p x_{pi}. \tag{4.4.1}$$

The optimal coefficients $\boldsymbol{\alpha}^*$ is fitted with ordinary least squares (OLS), a more detailed explanation of this calculation is found in Section 3.2.2. The vector, $\boldsymbol{x}$ will vary for each portfolio constituent, and be determined by backward stepwise regression for dimensionality reduction, which is the same technique Enke and Thawornwong (2005) use for their linear regression benchmark. In a nutshell, the technique starts with regression of all of the inputs which were fed into the PCA. The coefficient which has the lowest t-static will be removed, and a new regression will be fitted. These steps will be repeated until all of the inputs $\boldsymbol{x}$ are statistically significant, i.e. has $|t - static| > 1.96$, which corresponds to a 95 % confidence interval for the variable.

Second, to evaluate the accuracy of the model, in relation to economic theory, the CAPM model will be constructed as benchmark. The model assumed to be the most relevant determining the expected return, defined as

$$\hat{\mu}_{b2}^{(q)} = r_f + \beta(r_M - r_f) \tag{4.4.2}$$

where β is calculated through regression, r_M, i.e. return of the market portfolio, is set to the arithmetic historical mean of MSCI World, and r_f, i.e. risk-free return, is set to STIBOR3M. Using CAPM as a benchmark is relevant due to its origin from EMH. As CAPM will be used as input in the training of the neural networks, the first two year's CAPM level will be estimated by future estimates of β and r_M. This is to not loose any data for training, and as it only affects the training of the model, it will not bias the evaluation of the model.

Both the OLS and CAPM will be estimated using all available data, instead of using the rolling window technique.

The main evaluation metric to measure the performance of the neural networks vs the benchmark, will be MAE defined as

$$MAE = \frac{1}{T}\sum_{i_1}^{T} |y_t - \hat{y}_t|. \tag{4.4.3}$$

The MSE used as objective function is neglected to the favor of MAE since in this context we do not need the properties of having a function with continuous derivative and that treats errors differently depending on their magnitude, the absolute deviation is of highest interest.

The SIGN will also be used to tell the probability of estimating in the right direction and is defined as

$$SIGN = \frac{1}{T}\sum_{i_1}^{T} z_t, \quad \text{where} \quad z_t = \begin{cases} 1 & \text{if} \quad y_t\hat{y}_t > 0 \\ 0 & \text{otherwise} \end{cases} . \tag{4.4.4}$$

Further, this is a metric that can foster discussion of whether EMH holds or not. The EMH implies that creating a model that can estimate the expected return with a SIGN higher than 50 % is not possible. To draw any conclusion whether the neural networks can estimate the expected return better than the benchmarks the Diebold and Mariano (1995) test will be conducted using a one-sided alternative hypothesis. This test will be conducted for every portfolio constituent to evaluate whether the findings are consistent over different assets. The Diebold and Mariano (1995) test requires less assumptions than some of the other similar tests found in previous research, which increases the validity of possible findings. The significance level that the null hypothesis can be rejected on for each of the return estimations will be studied as well. See Section 3.8.1 for a detailed description of the test.

4.4.2 Phase 2

The second evaluation phase will comprise the project at large and will be evaluated from two perspectives. First, as with the previous phase, relevant metrics will be used to compare different benchmarks where in this case the benchmarks consists of different portfolio strategies. Secondly, our portfolio optimization will be used with $\hat{\mu}_{b2}^{(q)}$ since there is not only of interest to evaluate the accuracy of our estimation of the expected value, but also how well it fits our portfolio optimization model. The most accurate estimation does not necessarily imply the best return in the portfolio optimization phase. This phase will be evaluated using weekly portfolio returns.

The first metric that will be used to evaluate the total portfolio performance will be Jensen's alpha, α_p which gives us the excess return over the market portfolio. Also, the regression will give us the systematic risk β_P. The Jensen's alpha will be calculated through linear regression fitted with least squares, and is defined as

$$r_{P,t} - r_{f,t} = \alpha_P + \beta_P(r_{M,t} - r_{f,t}) \tag{4.4.5}$$

where the return on the market portfolio $r_{M,t}$ at time t will be estimated with the MSCI World Index, which is one of the portfolio constituents. As previously mentioned, $r_{f,t}$ will be estimated with STIBOR3M. The portfolio return $r_{P,t}$ will be calculated according to the asset aggregation formula found in Section 3.3 as $r_{P,t} = \ln(\sum_{i=1}^{N} w_i e^{r_{i,t}})$ where w_i is the relative weight of asset i, and $r_{i,t}$ is the logarithmic return of asset i.

The statistical significance of the excess return, i.e. the alpha of the portfolio, will be validated through a Student's t-test with the null-hypothesis $H_0 : \alpha_p = 0$ against $H_1 : \alpha_p > 0$.

As risk adjusted metric the Sharpe ratio, S_r, will be used based on the assumption that the portfolio is the total holding and we are exposed to both systematic and unsystematic risk, something that the Treynor index does not take into account. The Sharpe ratio is defined as

$$S_r = \frac{r_P - r_f}{\sigma_P}. \tag{4.4.6}$$

where r_P and $r_f$will be the annualized mean returns of the portfolio and riskfree rate respectively, and σ_P will be the sample standard deviation of the portfolio returns.

In addition the the portfolios generated with $\hat{\mu}_{b2}^{(q)}$, a benchmark portfolio with equally weighted assets will be constructed. The equally weighted portfolio will consist of the same assets as the optimized portfolio, and be equally weighted at the initialization, but not rebalanced during the course of time, i.e. buy-and-hold. Also, a mean variance portfolio with quarterly weight re-allocations will be constructed as benchmark according to the model

$$\max_{\substack{\mathbf{1}^T\mathbf{w}+w_{r_f}=1 \\ \mathbf{w},w_{r_f}\geq 0}} \quad \boldsymbol{\mu}^T\mathbf{w} + \frac{\gamma_{MV}-1}{2}\mathbf{w}^T\mathbf{C}\mathbf{w} + w_{r_f}r_f \tag{4.4.7}$$

where $\boldsymbol{\mu}$ will be estimated with $\hat{\boldsymbol{\mu}}_{b2}^{(q)}$, and $\mathbf{C}$ will be estimated with the sample covariance. The riskfree rate r_f will be estimated with STIBOR3M. To make a fair comparison of the stochastic programming optimized portfolio, the same constraints will be invoked to the mean variance portfolio, i.e. no shorting and no borrowing, as is defined in (4.4.7). The risk aversion parameter γ_{MV} will be set in accordance to the risk aversion level in the stochastic programming model, and will as such yield seven portfolios. The mean variance portfolio will be optimized in Matlab.

4.4.3 Summary of evaluation

In Table 4.4.1 there is a summary of the evaluation metrics, tests and benchmarks to be used in the evaluation of the model in this thesis.

Table 4.4.1: Summary table of the evaluations

	Metrics	**Statistical tests**	**Benchmarks**
Phase 1	MAE SIGN	Diebold-Mariano	OLS CAPM
Phase 2	α_P S_r	t-test of α_P	SP with $\hat{\mu}_{b2}^{(q)}$ Equally weighted portfolio Mean variance portfolio

Chapter 5

Results & Analyses

This chapter presents the results from the first and second phase of the thesis respectively. Each of the phases are evaluated and analyzed in relation to the objective of the thesis.

5.1 Phase 1: Neural Network

The first step of phase 1 was to perform extensive testing on different error functions and hyper-parameters. With these results we got an understanding of the dynamics of the model to determine an error function and set the limits for the random search. Thus, the results from these tests will be presented as a rationale for determining the error function and the limits for the random search. Afterwards, the result of the predictions in the test set will be provided and analyzed.

To keep the readability of the report, based on the numerous predicted time series, we will mainly present in-depth analysis from the neural network designed for the swedish equity index for large cap OMXS30. This index's neural network will act as an example for logical reasoning that apply to the other time series as well. The actual results for all of the time series are available in the Appendices.

5.1.1 Error function

Different error functions will penalize the error between the outcome of the neural network and the targeted output differently, and thus the statistical properties of the predictions will vary according to this choice. In the following test an understanding of how different error functions affect the prediction was gathered to further motivate this decision. As described in Section 4.2.7 the smallest possible error of the predictions for a specific time series with a variance that is below the predefined level is of interest.

Exponents

The behaviour of the predictions with an error function defined as the sum of the errors with different exponents $b \in \{0.1, 0.5, 1, 2, 6, 26\}$ have been evaluated in the MSS. According to the mathematical theory an increased exponent penalizes greater

errors at the cost of ignoring smaller errors, which can be seen in the histograms in Appendix E.1. This behaviour might be seen as desirable due to its great fit, but a great fit in the MSS might imply an overfitted network with reduced generalization ability. Further, the empirical results indicate that minimizing greater errors increases the variance of the predictions.

An exponent below one has shown to penalize the errors in the opposite way, which aligns with the intuition. The hypothesis was to approach the mean return during the period while also reducing the variance but the test shows that the most common outcomes of the target returns are captured instead. This could potentially be problematic when the most common outcomes deviate from the mean, which is exemplified in Figure 5.1.1.

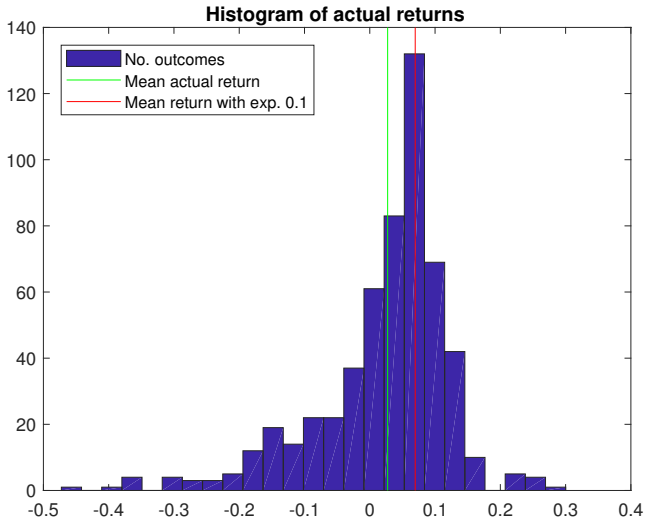

Figure 5.1.1: Histogram of the actual returns for OMX Index in the MSS

None of the exponents smaller than one or greater than two have shown desirable properties. The estimations for $b \in \{1, 2\}$ show the most promising behaviour, and together with previous researchers' choice of objective function, $b = 2$ is utilized in our model. What further incentivize this choice is that $b = 2$ grants the network with an error function, unlike $b = 1$, that is a continuous, differentiable function, useful for gradient descent optimization. Also, it is an unbiased estimator of the expected value, see Section 3.2.1.

Figure 5.1.2 presents the estimations in the MSS against the targeted output using $b = 2$ for OMX. The volatility in the MSS is approximately 6% and the mean return is 4%. This means that with a significance level of 5 % the estimations of the return is approximately[1] $\hat{\mu}^{(q)} \in [-10\%, 16\%]$. This would likely have a bad performance in the test set, as well as causing vast re-allocations in the portfolio optimization phase. Thus

[1]This would hold if the predictions were normally distributed with expected value 4 % and standard deviation 6 %. Whether or not that is true is not important, this is just to approximate a confidence interval for the estimations.

variance reduction via CAPM regularization was decided to be used with coefficient estimated according to the test described in the section below.

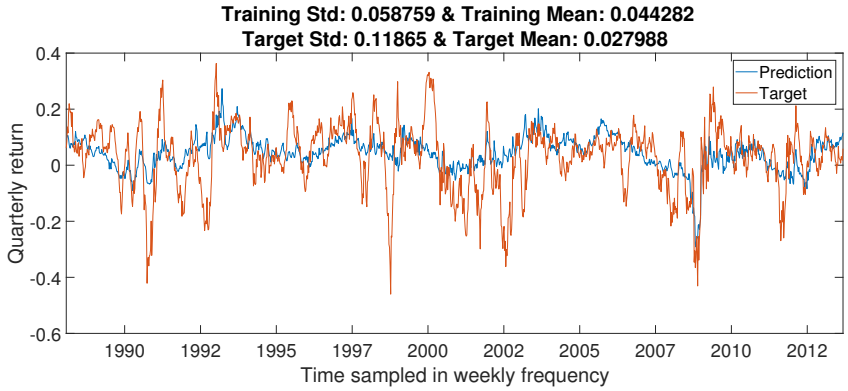

Figure 5.1.2: Plot of estimations in the MSS with SSE error function

CAPM regularization

In the following test, with the modified error functions from (4.2.12) and (4.2.13), we evaluated how the dynamics of the predictions varied for different λ in the MSS. The regularization coefficients were set to λ_{CAPM}, $\lambda_{RW} \in \{0, 0.5, 1, 1.5, 2, 4\}$ and the resulting plots and histograms for the OMX CAPM regularization test can be found in Appendix E.2. First, the actual volatility, σ_{Actual} of the target data was determined. Second, the volatility of the estimates, σ_{NN} given the different coefficients were put in relation to the targeted volatility, σ_{Target}, defined as, $\sigma_{Target} = 0.15\sigma_{Actual}$. Due to the fact that the volatility of CAPM, σ_{CAPM}, was as low as 0.3% for OMX the outcome of this test was as expected; an increased value of λ_{CAPM} resulted in the desired property of reduced volatility.

The decision model to determine λ_{CAPM} or λ_{RW} is exemplified with the OMX Index. Through observation of the characteristics provided in Table 5.1.1, which represent the results from the analysis, the coefficient could be set.

Table 5.1.1: Analysis of the volatility for different λ_{CAPM} for the error function of the neural network designed to the OMX index

$\boldsymbol{\sigma_{Actual}}$	$\boldsymbol{\sigma_{Target}}$	$\boldsymbol{\sigma_{CAPM}}$
0.119	0.018	0.003
$\boldsymbol{\lambda_{CAPM}}$	$\boldsymbol{\sigma_{NN}}$	Below σ_{Target}?
0	0.036	No
0.5	0.026	No
1	0.014	Yes
1.5	0.013	Yes
2	0.012	Yes
4	0.011	Yes

To sum up; the error function for the neural network representing OMX, with $N = 1337$ available training examples, was set to

$$J_{OMX}(\mathbf{W}) = \sum_{m_t=1}^{1337} (y_{m_t} - \hat{y}_{m_t})^2 + 1 \sum_{m_t=1}^{1337} (\hat{\mu}_{CAPM,m_t} - r_{f,m_t} - \hat{y}_{m_t})^2 \tag{5.1.1}$$

5.1.2 Hyper-parameters

The hyper-parameters of the final network have been chosen through optimization with parameters chosen from the random search method. But before, limits for the random search method was determined.

Ideally, the number of random searches should be as high as possible, as a higher number increases the chance of finding a good solution. But computational time is a limiting factor. Too many random searches would be very time consuming. Based on testing of computational time, the number of random searches was set to 40, which can be put in contrast to the number of possible grid searches of 560 for the choices of limits that have been done.

How the random searches should be distributed between the different hyper-parameters and the sample space for each of them is not obvious from the literature. This leaves the designer to understand the variation in the dynamic of the networks, for each choice of hyper-parameter. Thus, tests have been run to evaluate what choices of hyper-parameters that might be the optimal choices for the final network, and the limits have been set accordingly. The total time for optimizing the 15 neural networks was 9h:33min[2].

Explanation degree

Different number of inputs to the network were evaluated i.e. different degrees of explanation, f, of the PCA were tested, remember that the inputs to the network is the principal components of the PCA. According to the theory, the number of inputs affects the performance of the network. Consequently, a test of how many eigenvectors, from the PCA, that is required to achieve different degrees of explanation was performed. The test was performed on the OMX Index. According to Figure 5.1.3 an increased f for high values increases the number of eigenvectors more compared to an increase in the lower levels. For example an increase of f from 0.90 to 0.95 increases the number of inputs with 4 while increasing from 0.995 to 1 increases the inputs with 7. To allow the model selection algorithm to test different number of relevant inputs, the interval of f to utilize random search over varied from values between 0.9 to 1. Based on this test there will be a higher frequency of possible f closer to 1, hence the possible f is left to be randomly searched in the set of $f \in \{0.90, 0.95, 0.975, 0.995, 1\}$.

[2]Computational time of the model is dependant on what computer that is used and will vary according to its processor power.

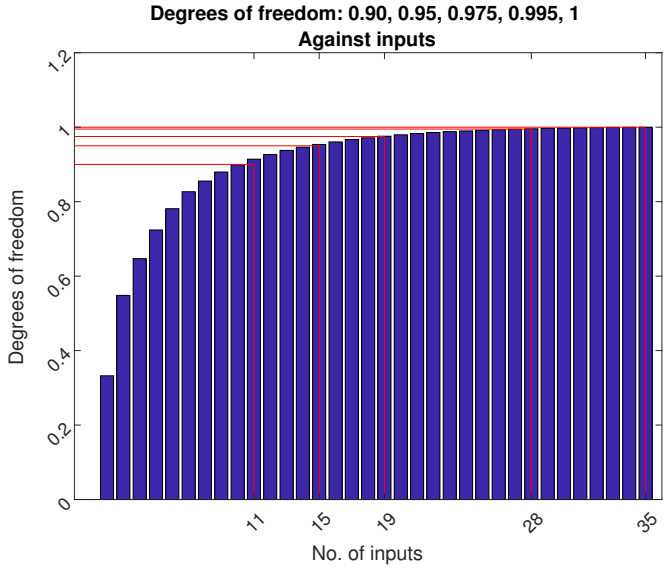

Figure 5.1.3: The cumulative degrees of explanation for different number of eigenvectors

To potentially further reduce this hyper-parameter a test of which f that performs the lowest MSE in the validation set for different architectures was performed. First, this test was done on the OMX Index. The left hand side of Figure 5.1.4 shows that the lower values of the set of f were irrelevant. To ensure the strength of the test further analysis was conducted. According to the theory the size of the architecture can affect the number of inputs since the number of free parameters should not exceed a specific upper limit. But, no relations that proves this theory were found. Another theory claims that the number of free parameters times 10 should not exceed the training data, hence indices with less available data should limit the number of inputs. Consequently, the same test was performed on a shorter index, namely NOMXCRTR[3]. The results are provided in Figure 5.1.4.

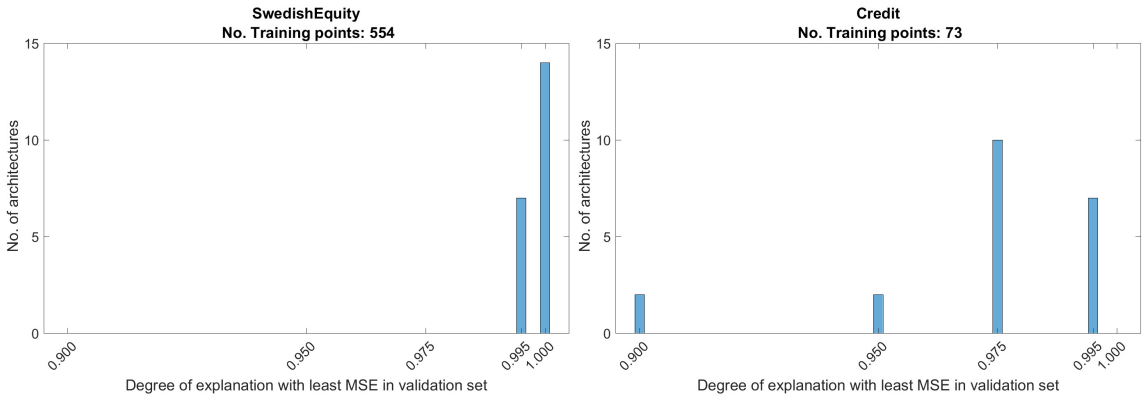

Figure 5.1.4: Relation between the degrees of freedom and size of training plus validation set

[3]This index was only used for this test

Whether the theory holds or not, lower f might be of interest and the set of f will remain the same as previous and set according to Table 5.1.2.

Table 5.1.2: Search space for the random search for f

Hyper-parameter	**Possible values**
Explanation degree f	$f \in \mathbb{R}: \ f \in \{0.90,\ 0.95,\ 0.975,\ 0.995,\ 1\}$

Number of hidden layers and neurons

To determine the limits for the random search a test was conducted. This was designed to find the best architecture from multiple randomized architectures of different sizes, layers as well as neurons. To simulate the test set performance as good as possible the performance metric for each architecture was the MSE from the validation set. All architectures had the same initial conditions, number of cross validations and data partitioning to avoid random results. The tests were evaluated on a longer and a shorter time series to test whether the theory that claims that with restricted amount of data a smaller network should be used. No obvious relationships were found that supports the theory though, and therefore the limits were set to be consistent for each time series. The choice of limits for the neurons in every hidden layer was based on the architectures of the best networks in the test to minimize the risk that the random search misses relevant architectures. The limits for layers and hidden neurons is summarized in 5.1.3.

Table 5.1.3: Search space for the number of hidden layers, and the number of neurons in each hidden layer for the random search

Hyper-parameter	**Possible values**
No. hidden layers L_h	$L_h \in \mathbb{Z}: \ L \in [1, 3]$
No. neurons if one hidden layer M_2	$M_2 \in \mathbb{Z}: M_2 \in [1, 16]$
No. neurons if two hidden layers M_2, M_3	$M_2,\ M_3 \in \mathbb{Z}: \ M_2 \in [1, 12],\ M_3 \in [1, 5]$
No. neurons if three hidden layers M_2, M_3, M_4	$M_2,\ M_3,\ M_4 \in \mathbb{Z}: \ M_2 \in [1, 9],\ M_3 \in [1, 4],\ M_4 = 1$

5.1.3 Results from the optimized neural networks

The design of the resulting artificial neural networks determined by the MSA and its performance will be presented in this section. The generated ANNs varied vastly in terms of hyper-parameters for each time series, see Appendix F.

The number of potential inputs for OMX Index was 32, and the relevance analysis reduced it to 19. Further, the random search ended up with an explanation degree of 1.000 leading to the use of all 19 eigenvectors from the PCA. Together with CAPM

a total of 20 inputs were fed to the network. The architecture chosen consisted of 1 hidden layer with 5 hidden neurons. The architecture is summarized in Table 5.1.4.

Table 5.1.4: Results from the hyper-parameter optimization

Hyper-parameter	**Optimal values**
Explanation degree f	1.000
No. inputs M_1	20
No. hidden layers L_h	1
No. neurons first hidden layer M_2	5
No. outputs M_3	1
Resulting architecture	(20-5-1)

The volatility of the estimation in the MSS was 0.014 (CAPM 0.003) and the volatility in the test set was 0.0045 (CAPM 0.0003) compared to the actual volatility of 0.119 and 0.06 respectively. The mean of the estimations in the MSS was 0.023 (CAPM 0.020) compared to the actual mean during the same period of 0.028. The mean of the estimations in the test set was 0.032 (CAPM 0.016) compared to the actual mean during the same period of 0.019. The results are summarized in Table 5.1.5.

Table 5.1.5: Summary of results from the OMX-network vs. CAPM and actual returns

	MSS			**Test Set**		
Parameter	ANN	CAPM	Actual	ANN	CAPM	Actual
σ	0.014	0.003	0.119	0.005	0.000	0.060
$\bar{\mu}$	0.023	0.020	0.028	0.032	0.016	0.019

The estimations are seemingly closer to the mean both in the MSS but not in the test set, and for both cases to a higher volatility. In Figure 5.1.5 the predictions of the ANN is plotted compared with the actual returns and the CAPM predictions. Plots for the estimations for every index in the test set can be found in Appendix G.

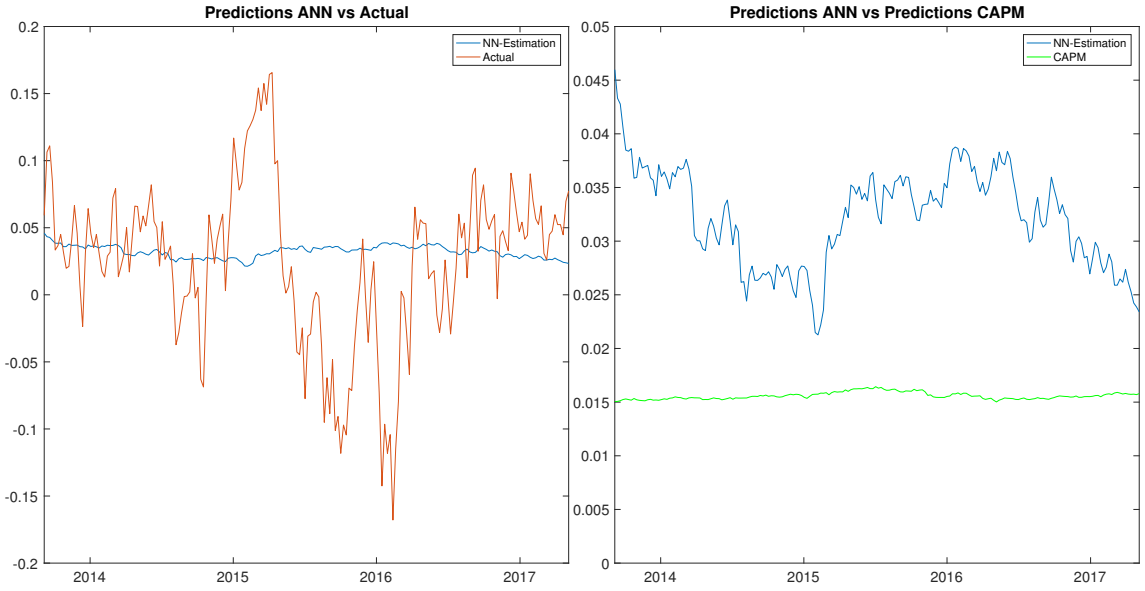

Figure 5.1.5: Left: Test set predictions of ANN vs actual returns. Right: Test set predictions of ANN vs CAPM

5.1.4 Evaluation

To evaluate whether our neural network model is suitable for estimating the expected return, the performance in the test set is compared to other available models, namely CAPM for the indices and RW for the FX-rates. Some of the existing research papers in this field fail to compare the results to relevant methods used for estimating the expected return in finance, instead linear regression (OLS) is a common benchmark to conclude if a model is successful or not.

Benchmarks

In Table 5.1.6 a comparison of the MAE and SIGN metrics for the models respectively can be found. For the fifteen time series, the ANN produced lower MAE than CAPM/RW only for four time series, the two equity indices OMX and MXWO, the interest rate index RXBO and the FX spot rate of JPY/SEK. For most of the other indices, the results are similar or slightly worse than those of CAPM/RW. However, there seems to be no evidence that our model could be better than CAPM/RW with perspective of MAE. The SIGN metric. i.e. to what rate the prediction is in the correct direction, is for the indices dominated by CAPM. Though, the importance of this metric is questioned.

Comparing the results to OLS, twelve out of the fifteen time series were predicted with a lower MAE by the ANN. This indicates that our ANN model potentially is better than an OLS model. In Section 3.9 there is a summary of the paper of Enke and Thawornwong (2005). The authors observe that they produce a lower RMSE than the OLS and a SIGN of 0.68, and argue that their results are successful without investigating the statistical significance of their findings, nor compare it with econometric

models like CAPM. It is hard to tell whether their results generalize to other indices than S&P500 as the study only considers that one index. However, considering the comparable equity indices with perspective of MAE in our case and RMSE in their as well as the level of SIGN the results seem to conform.

The results of this study are hard to compare to Qi and Zhang (2001) estimation of S&P500 as they only benchmarks against AR and RW models. However, they only slightly beat the RW model, and as our results for the equity indices slightly beat those of CAPM, the results are considered to be at least in line with Qi and Zhang (2001).

Considering the FX-rates, our results for the MAE is quite similar to the benchmarks, i.e. the RW and the OLS, which is a worse result than those of Hann and Steurer (1996). This might be explained by different methodologies determining inputs to the ANN, they utilized an econometric monetary model not comprised by our model.

Table 5.1.6: Out-of-sample performance metrics for the ANN, CAPM and OLS for every index. The lowest MAE and the highest SIGN for every index respectively is marked with bold font. At the bottom, the number of times ANN was better than CAPM/RW and OLS respectively is presented.

	MAE (*100)			**SIGN**		
Index	ANN	CAPM	OLS	ANN	CAPM	OLS
OMX	**4.605**	4.732	6.480	0.719	0.719	0.354
MXWO	**4.686**	4.781	8.300	0.776	0.776	0.250
RXBO	**0.92**	1.001	3.412	0.630	**0.641**	0.255
RXRE	1.674	**1.654**	8.987	**0.542**	0.490	0.339
RXVX	0.132	**0.051**	0.163	0.766	**0.969**	0.521
HE00	1.856	**1.795**	4.829	0.786	0.786	0.458
LEGATRUU	3.468	**3.118**	5.170	0.245	**0.641**	0.464
JGENBDUU	4.227	**4.127**	6.829	0.630	**0.677**	0.359
MXWO0RE	5.309	**5.254**	8.666	**0.635**	0.630	0.396
BCOMTR	5.293	**5.075**	7.106	0.453	0.453	**0.51**
HFRXM	5.112	**4.568**	8.200	0.417	**0.583**	0.453
HFRXAR	4.272	3.749	**2.69**	0.406	0.693	**0.714**
FX-rates		**RW**			**RW**	
JPYSEK	**3.969**	3.984	4.402	**0.568**	0.000	0.411
EURSEK	1.938	**1.67**	1.886	0.375	0.000	**0.432**
USDSEK	4.240	**3.738**	4.191	**0.417**	0.000	0.359
ANN vs.	-	4	12	-	4	9

Statistical test

Also, the Diebold and Mariano (1995) test was conducted, and the results are presented in Table 5.1.7. The use of this model proves to be relevant in this context, since the prediction errors seem to be non-gaussian and at times with non-zero mean. None of the time series where the ANNs had a lower MAE than CAPM had statistically

significant lower prediction errors. As such, the result can be based on merely luck. Looking at the fifteen time series, the P-value has an obvious randomness. What can also be noted is that the P-value for none of the indices is above 0.950, which means that the null hypothesis, given a reformulated alternative hypothesis that the CAPM estimates produce significantly better estimates than the ANNs, can never be rejected. Conclusively, a claim that the either the ANN model or CAPM/RW would provide better estimations than the other can not be drawn.

Looking at the results vs. OLS, two of the time series is shown to be statistically significant better than the OLS, namely the RXBO Index[4] and the RXRE Index[5]. However, for the other time series (where ANN is better than OLS) it is impossible to draw any consistent statistically significant conclusions whether the ANN predictions are better or not. Ideally to validate with consistency whether the ANN outperforms OLS one would need longer time series for these tests than those considered in this thesis, as to have sufficient amount of training data, i.e. a longer test data set than 192 data points. To conclude statistical significance with a limited number of data points is nearly impossible. Whether it is relevant or not to show stochastic dominance against an OLS model in estimating the expected return in finance is another matter, since no evidence is provided in the finance literature for such models.

5.1.5 Sensitivity analysis

A sensitivity analysis was conducted with the aim to unravel the impact of changing the error function (5.1.1). This was considered the most important parameter, and a sensitivity analysis of the other parameters in the model was consequently not conducted. Testing other inputs would be of interest though, but the scope of such analysis exceeds the scope of the sensitivity analysis.

Using the SSE error function

The ANN model was run with the SSE error function, which basically means that the CAPM regularization coefficients were decreased to zero, i.e. $\lambda_{CAPM} = 0$. The results were less promising. Only 1 out of the 15 estimations had a lower MAE than CAPM. Although, the SSE error function predictions did beat the OLS predictions in 9 out of the 15 time series, it is hard to evaluate whether ANNs with this error function is better than the models from the other studies due to the low number of indices in those studies. However, this error function seem to impair the results due to lack of generalization ability and the hypothesis made in Section 5.1.1 that a SSE error function overfits the training data seems to hold. The results from this test is summarized as:

- 1 out of the 15 ANNs had lower MAE than CAPM/RW
- 9 out of the 15 ANNs had lower MAE than OLS

[4]OMRX Total Bond Index, an interest rate index
[5]OMRX Real Return Index, an inflation index

Table 5.1.7: Diebold-Mariano test for forecast error (MSE) unequality. ANN estimations tested versus CAPM and OLS estimations respectively. If the test static had a P-value < 0.05 the null hypothesis of estimation error equality was rejected, and the ANN estimation was concluded to provide a statistically significant better estimation.

	vs CAPM		**vs OLS**	
Index	H_0 rejected?	P-Value	H_0 rejected?	P-Value
OMX	no	0.469	no	0.307
MXWO	no	0.364	no	0.227
RXBO	no	0.294	**yes**	**0.027**
RXRE	no	0.538	**yes**	**0.013**
RXVX	no	0.843	no	0.401
HE00	no	0.515	no	0.223
LEGATRUU	no	0.767	no	0.273
JGENBDUU	no	0.667	no	0.281
MXWO0RE	no	0.529	no	0.236
BCOMTR	no	0.588	no	0.319
HFRXM	no	0.617	no	0.247
HFRXAR	no	0.645	no	0.681
FX-rates	**vs RW**		**vs RW**	
JPYSEK	no	0.486	no	0.365
EURSEK	no	0.644	no	0.517
USDSEK	no	0.610	no	0.513
Times rejected	0	-	2	-

Changing the CAPM regularization coefficient

The CAPM regularization coefficient was decreased such that the network predicted with volatility corresponding to 10 % of the original volatility in the MSS, i.e. $\sigma_{Target} = 0.10\sigma_{Actual}$. This new requirement on the volatility resulted in higher CAPM regularization coefficients for each index. The results were slightly better, 7 out of the 15 estimations had a lower MAE than CAPM/RW compared to 4 with the existing error function. Regarding the OLS model 14 out of 15 ANNs produced predictions that outperformed the OLS model, compared to 12 with the existing error function.

- 7 out of the 15 ANNs had lower MAE than CAPM/RW
- 14 out of the 15 ANNs had lower MAE than OLS

The performance in the test set seems to increase as the CAPM regularization coefficient is increased. Figure 5.1.6 shows a comparison of the ANN predictions and the CAPM predictions in the test set for OMX Index.

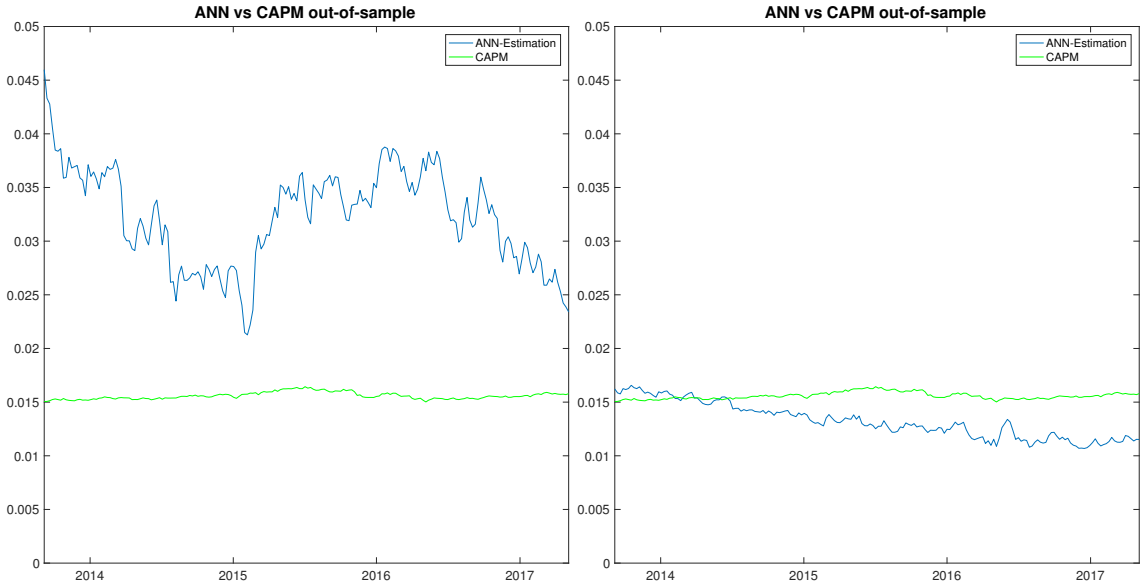

Figure 5.1.6: OMX Index predictions. Left: Out-of-sample predictions of ANN with lower regularization coefficient ($\lambda_{CAPM} = 1$) vs CAPM predictions. Right: Out-of-sample predictions of ANN with higher regularization coefficient ($\lambda_{CAPM} = 5$) vs CAPM predictions

Intuitively, the higher the CAPM regularization coefficient, the more similar the ANN predictions are to CAPM. Also, the volatility of the estimates decreases. In this context, one could dispute the relevance of using the more complex ANN in favor of CAPM. Instead of estimating according to its own paradigms, it follows another model. As such, the final model is more of a tweak of CAPM than a model by its own and this to a cost of increasing the model complexity tremendously. In order to develop an alternative model through ANN with estimations according to its own dynamic that significantly is competitive to CAPM, when it comes to MAE, other possible modifications has to be investigated.

Nevertheless, extension of CAPM or independent model, penalize errors in relation to economic models as a part of the ANN optimization seem to be a possibly promising, further research area if the purpose is to end up with more accurate predictions.

5.1.6 Test for model convergence

A crucial part of a predictive model is its robustness. Ideally, given the same conditions, one wish to have the same estimation for the same period for every run, i.e. that the model converges to the same solution. A model that converges to the same solution for a non-convex problem can be hard to design. Hence, it is desirable that the predictions at least end up within a narrow interval.

In this model there are two factors that adds stochastic to the model. One factor is the random search method that tries to find the optimal set of hyper-parameters. By nature it adds stochastic to the outcome from the model, as the identification of poten-

tially good hyper-parameters are random. The second factor is the initial conditions. Since the optimization problem is non-convex, the initial conditions are randomized each run, and as such, the model risk to converge to different local minimas. Furthermore, the cross validation method enhances the stochastic of the results. This is due to the fact that the final neural network that is chosen by the MSA will be trained on different data from run to run. Based on the training examples, the resulting weights might differ quite a bit. The random permutation of the data will not add stochastic to our results, as the permutation is kept the same for all runs.

Consequently, tests were carried out to study the convergence of the training of the network. The aim was to find out whether one trained network predicts, for the upcoming quarter, the same expected return, or at least similar, as another trained network, given the same model specifications.

Although using 9 initial conditions, which is in the interval proposed by Glorot and Bengio (2010), the test converged to different weight solutions for different runs, which led to different predictions in the test set. This can be seen in Figure 5.1.7.

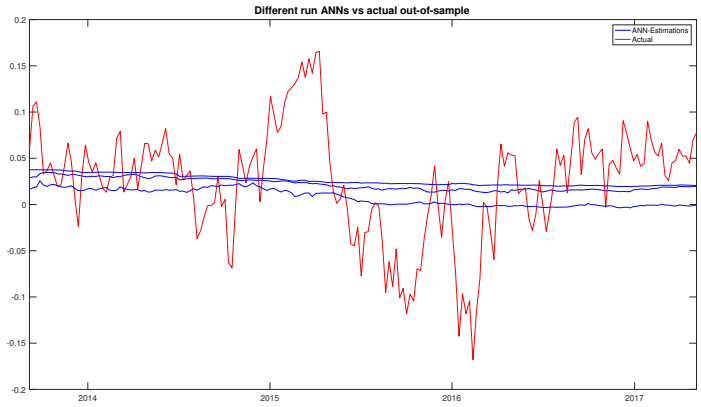

Figure 5.1.7: Estimated expected returns for three different runs of the ANN model vs the actual returns

Ideally to create a deterministic solution one must overcome the problem of determining the optimal hyper-parameters. This could be done by using grid search over a infinite large interval of the hyper-parameters. To overcome the problem of non-convex optimization in the training phase an infinite number of initial conditions could be implemented. But the computational power for such a model is obviously unreasonable in this thesis.

Enke and Thawornwong (2005) utilized cross validation and the ensemble averaging technique described in Section 3.4.4 which potentially would have made the model more stable. But, it could be argued that averaging several predictions that are different is not a good idea, and instead a model for generating good forecast in the first place would be better, thus we neglected this method.

However, this makes the proposed model, given this specifications set unfeasible for use for Söderberg & Partners. There is obviously a lack of robustness in the model when training the network, which decreases the reliability of the predictions. Further

research has to be done to increase the robustness of the model where the number of initial conditions, the cross validation method and random search methodology is hypothetically influencers of the model's convergence capability and thus suggested modification areas.

5.2 Phase 2: Portfolio optimization

This phase branches into two steps. First, stochastic processes were estimated in the MSS and assigned for each one of the portfolio constituents. Second, the NN-portfolios and the benchmarking portfolios were optimized and run in the test set. In contrast to phase 1 most of the involving parameters could prespecified, besides the stochastic processes, but what is left to unravel is the number of scenarios used in the model to describe possible returns from the processes.

This section will provide the results from the second phase of the thesis and the model in it is entirety. Further, these results will be analyzed with regards to the chosen performance metrics and subsequently put in comparison to the benchmarks.

5.2.1 Results from estimated stochastic processes

For each index, GBM, student's t and GARCH-Poisson processes were estimated. Based on the estimated parameters, quarterly returns for every index at weekly frequency were transformed via the inversion principle and plotted in QQ-plots. The QQ-plots did generally not show any linearity. For some of the indices, the untransformed log-returns unexpectedly yielded a straighter line than any of the tested processes. One hypothetical reason is the fact that the tails are less present on longer time horizons as indicated by Cont (2001). Another is that our model comprise the expected return from the ANNs, for which those are varying through time and capture parts of the tails. Combined with a time varying volatility this causes an unwanted leverage effect. Based on the bad fit, a fourth process, GBM with fix volatility was tested with better results than the other processes for some of the indices, see an example in Figure 5.2.1.

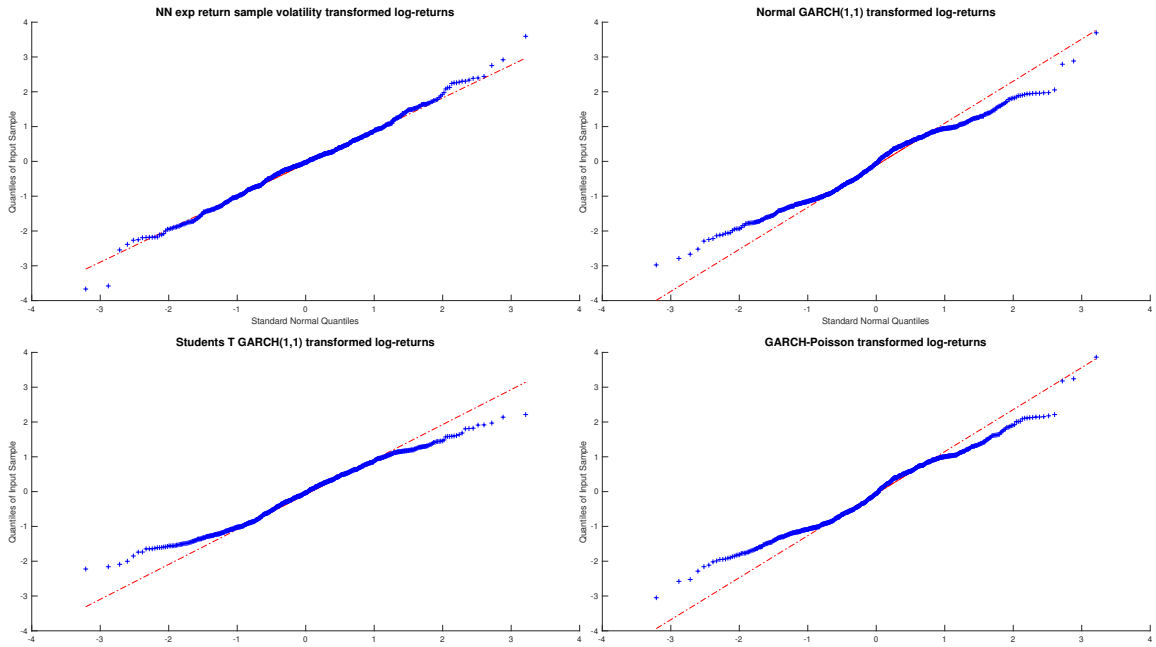

Figure 5.2.1: QQ-plots for HFRXAR Index. Upper left: GBM transformed log-returns. Upper right: GBM GARCH(1,1) transformed log-returns. Lower left: Student's T GARCH(1,1) transformed log-returns. Lower right: GARCH-Poisson transformed log-returns

In Table 5.2.1 the processes chosen to model the portfolio constituents in the test set are presented. The same methodology was conducted when determining processes for the benchmark portfolio, SP with CAPM as estimator of the expected return. For CAPM, GBM with sample volatility was chosen for 7 indices, Student's T GARCH(1,1) was chosen for 4 indices, and GARCH-Poisson was chosen for 1 index.

Table 5.2.1: Chosen processes for each index.

Process	**Indices**	**Count**
GBM sample volatility	MXWO, RXBO, RXRE, RXVX JGENBDUU, BCOMTR, HFRXM, HFRXAR	8
GBM GARCH(1,1)	-	
Student's T GARCH(1,1)	OMX, HE00, LEGATRUU, MXWO0RE	4
GARCH-Poisson	-	

5.2.2 Results from portfolio optimization

From tests of how different number of scenarios for the return at each re-allocation affect the computational time in this model, the number of scenarios was set to 1 000. The results from the portfolio optimization for the seven risk levels are presented in Table 5.2.2. The annualized return for the SP portfolios using neural network for estimating the expected return (SPNN) is higher for the two most risk averse profiles

than those of the SP portfolios with CAPM estimating expected return (SPCA). Remarkable though is that the same risk profiles are the only profiles that have higher volatility than SPCA. Compared to the MV portfolios with CAPM estimated expected return, the SPNN is constantly beaten on the annualized return but through all the risk profiles the risk of SPNN is lower. Compared to the equally weighted portfolio, the return is higher for the four least risk averse and approximately equal for $\gamma = -7.7$. For the two most risk averse profiles, the SPNN is beaten. Noticeable from Table 5.2.2 is the strong relationship between the return and volatility. Regardless of model the most volatile portfolio seems to be generating the highest return. The Sharpe ratio left to analyze will be telling of how good the riskadjusted return was for the portfolios respectively, see Section 5.2.3.

In Figure 5.2.2 the wealth development of the portfolios for four risk levels is presented, $\gamma = -0.5, -2.99, -7.70, -27$.

Table 5.2.2: The results of the portfolios using stochastic programming with neural network estimated return (SPNN), stochastic programming with CAPM estimated return (SPCA), equally weighted portfolio (EqWe), and the mean variance portfolio (MV). The portfolio return and volatility are annualized.

	Return (R)				**Volatility ($\boldsymbol{\sigma}$)**			
Risk level ($\boldsymbol{\gamma}$)	SPNN	SPCA	MV	EqWe	SPNN	SPCA	MV	EqWe
-0.50	0.103	0.141	0.145	0.066	0.130	0.135	0.139	0.061
-1.46	0.091	0.136	0.145	0.066	0.115	0.134	0.139	0.061
-2.99	0.086	0.112	0.138	0.066	0.096	0.118	0.135	0.061
-5.29	0.082	0.090	0.118	0.066	0.076	0.093	0.115	0.061
-7.70	0.066	0.067	0.093	0.066	0.063	0.069	0.106	0.061
-11.20	0.055	0.049	0.074	0.066	0.053	0.049	0.101	0.061
-27	0.035	0.022	0.048	0.066	0.029	0.022	0.096	0.061
ANN vs.	-	2	0	4	-	5	7	2

External specifications

The two main properties desired, besides generating portfolios with a good risk-adjusted return, was to keep the re-allocations moderate while also holding a diversified portfolio. As such, analyses have been made on the volume of the total re-allocations and to what rate a diversified portfolio is held of the SPNN portfolios at each rebalancing. The sum of the absolute buy- and sell decisions divided by two is defined as the re-allocations at a time step, as such the maximum re-allocations for one step is 1, i.e. selling all previous holdings, and investing them in new holdings. Commonly this sum is denominated as the portfolio turnover. The mean of the re-allocations for each quarter varied between 0.19-0.27 between the risk profiles for the SPNN portfolios. The maximum single re-allocation was between 0.49-0.99 for the risk levels, i.e. almost the entire portfolio was re-allocated for one of the risk levels. This is highly

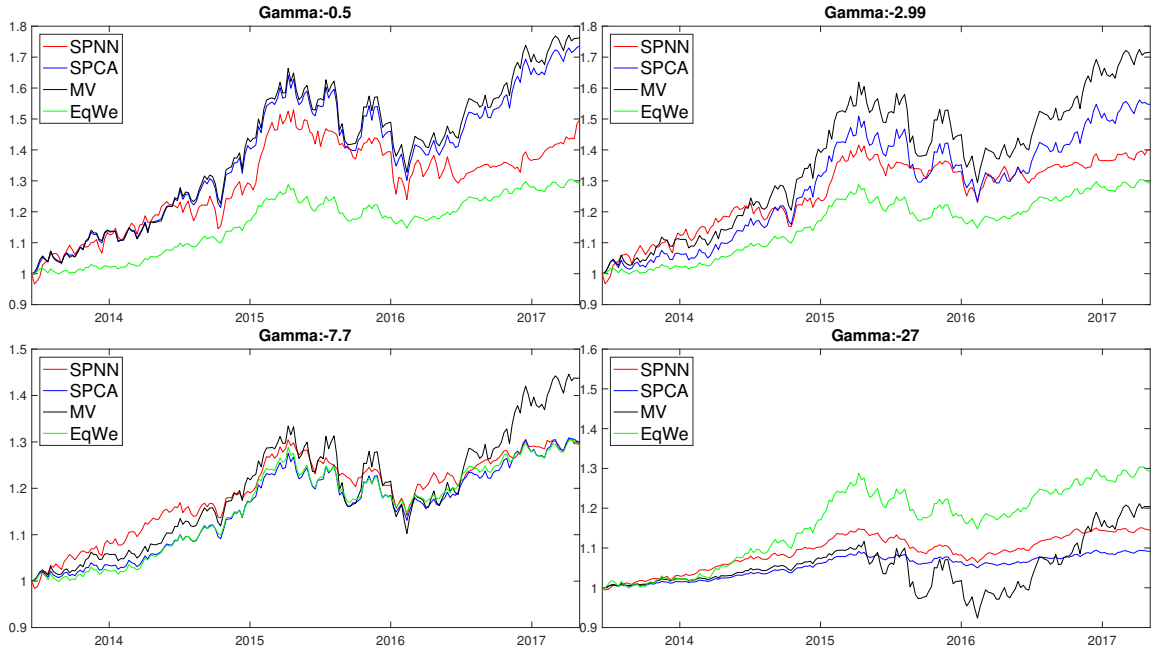

Figure 5.2.2: Development of the wealth given four investor risk levels, $\gamma = -0.5, -2.99, -7.70, -27$.

unlikely to be an acceptable level of a financial counseling company and violate the limits set by S&P, see Section 4.3.2.

To ensure that this phenomena does not depend on the SP portfolio optimization method used in this model one could compare the results of using CAPM as estimate in the SP versus the MV portfolios where CAPM is also utilized. The mean of the re-allocations using CAPM in the SP model varied between 0.00-0.05 for the portfolios, with a maximum of 0.01-0.15. The MV portfolios had a mean of re-allocations between 0.00-0.07, with a maximum of 0.00-0.80. As such, the SP model seems to lower the re-allocations, which is in line with the intuition, since the allocation costs are part of the objective function in the SP model.

It is obvious that the less volatile estimates of the expected return the less re-allocations, stemming from the fact that SP with CAPM led to lower re-allocations in the portfolio than the SP of ANN. Due to the fact that our estimates still has a lower volatility than the actual volatility there is a paradoxical relationship between accomplishing accurate estimates and keeping the re-allocations sufficiently low. One could of course increase the transaction costs in the objective function, but the implication of that would be that the full potential of a potentially accurately estimated expected return not can be seized.

One potential way of overcoming this issue is looking at how other portfolio management models as Black and Litterman (1992) can be applied together with neural network estimated expected returns.

The diversification of the portfolios at each time step is also an issue for the SPNN portfolio. For the higher risk levels, $\gamma > -6$, the portfolios seem to hold only 2-3 assets at each rebalancing, while at lower risk levels, $\gamma < -11$ the portfolios hold

approximately 4-6 assets. This tendency is however also noted in the benchmarks, SP with CAPM and MV, and as such the problems is not attributed to the neural network estimates, rather the portfolio optimization models themselves.

5.2.3 Evaluation

The evaluation metrics of phase 2 is presented in Table 5.2.3. The Sharpe ratio of the SPNN portfolios are for the three least risk averse portfolios lower than all of the three benchmarks, but for the four most risk adverse portfolios the Sharpe ratio is higher than both SPCA and MV. This means that even if the return for some of those portfolios were lower than the benchmarks, the volatility of the portfolios were to such an extent less that the risk adjusted return were higher. The SPNN portfolio did only have higher Sharpe ratio than the equally weighted portfolio for one of the risk levels, namely $\gamma = -27$. For all of the other risk levels, the equally weighted portfolio had the highest Sharpe ratio of all of the benchmarks.

The α of the SPNN portfolios are consistently higher than zero, which means that they during the period did create excess return on the market. The α for SPNN is however not statistically significant for any of the risk levels. Compared to SPCA and MV, the SPNN portfolios did consistently have a higher Jensen's alpha, even if the return was lower for most of the risk levels. This is partly based on the commonly lower volatility of the SPNN portfolios, but also that the portfolio returns were less correlated with the market returns. The α of the equally weighted portfolio is consistently higher than the α of the other portfolios. The metrics are significantly higher than zero at confidence level 0.03, at which level it can be statistically concluded that the equally weighted portfolio created excess return on the market during the period investigated. The α of 0.061 is almost as high as the actual return 0.066 during the period. This is based on the low β, which was only 0.02 for the period, based on low correlation of 0.06 between the portfolio returns and the market returns.

Table 5.2.3: Evaluation of portfolio optimization for using SPNN, SPCA, MV and EqWe. Evaluated using weekly portfolio returns. The annualized α with P-value of the t-test of α with null hypotehsis $\alpha = 0$ in parenthesis, and the annualized Sharpe ratio is presented.

	Jensen's α (P-value)				**Sharpe ratio**			
Risk level (γ)	SPNN	SPCA	MV	EqWe	SPNN	SPCA	MV	EqWe
-0.50	0.013(0.40)	0.001(0.45)	-0.000(0.94)	**0.061(0.03)**	0.79	1.04	1.04	**1.06**
-1.46	0.010(0.41)	-0.003(0.73)	-0.000(0.94)	**0.061(0.03)**	0.78	1.01	1.04	**1.06**
-2.99	0.017(0.31)	-0.01(0.82)	-0.002(0.68)	**0.061(0.03)**	0.89	0.94	1.02	**1.06**
-5.29	0.024(0.19)	-0.004(0.61)	-0.002(0.57)	**0.061(0.03)**	1.06	0.96	1.01	**1.06**
-7.70	0.016(0.23)	-0.002(0.59)	-0.014(0.83)	**0.061(0.03)**	1.03	0.96	0.86	**1.06**
-11.20	0.011(0.27)	-0.001(0.57)	-0.024(0.89)	**0.061(0.03)**	1.00	0.97	0.72	**1.06**
-27	0.010(0.15)	0.000(0.52)	-0.038(0.93)	**0.061(0.03)**	**1.13**	0.99	0.48	1.06
ANN vs.	-	7	7	0	-	4	4	1

5.3 Summary of results and analyses

The results and analyses can be summarized as follows:

Phase 1

• The ANNs did produce lower MAE than CAPM for 4 out of the 15 time series. None of the results were statistically significant	Section 5.1.4
• The ANNs did produce lower MAE than OLS for 12 out of the 15 time series. The results were statistically significant for two of the indices. Our results for the indices are similar to previous research	Section 5.1.4
• Regularizing the error function against CAPM did show more promising results than using the popular SSE error function	Section 5.1.5
• The model lacks reliability due to convergence to different solutions	Section 5.1.6

Phase 2

• The SP portfolios using neural networks for expected return estimation had higher Jensen's alpha than SPCA and MV on all risk levels, as well as higher Sharpe ratio on four out of seven risk levels. However, the equally weighted portfolio outperformed all of the benchmarks in terms of Jensen's alpha and almost all benchmarks in terms of Sharpe ratio.	Section 5.2.3
• Our ANN estimations were too volatile for the SP model, given the choice of transaction costs, based on the vast portfolio turnover to satisfy the external specifications set by S&P	Section 5.2.2

Chapter 6

Conclusions & Discussion

In this chapter conclusions from the empirical study will be presented, along with logical reasoning of how the method, results and conclusions helped to answer to the objective of the thesis. A discussion of the model and results in relation to previous research and economical theory, as well as potential shortcomings of the study is presented from which suggestions for future research is proposed. Lastly, ethical aspects of the study are discussed briefly.

6.1 Conclusions

Based on the results of this thesis, no conclusions regarding the possibility of accurately estimating expected return of broad asset class's financial times series with artificial neural networks can be drawn. Our model shows no evidence of outperforming CAPM nor RW as estimators of the expected return for the studied indices and FX-rates respectively. For some of the indices studied, the estimations of our model had lower MAE than CAPM, but no statistical significance was achieved. We can conclude that a hypothesis that our model could consistently generate better estimates than the reference methods seems highly unlikely. However, it is hard to generalize beyond the proposed model and the time series of the study. This is mainly based on that constructing neural networks incorporates a vast amount of design choices. Potential design choices that could impact the result, for better or for worse, include: input data selection, class of neural network, architecture, regularization technique and parameter choice, and more. There is a chance that a properly designed neural network model could improve the estimated expected return estimations in relation to widely used financial models for specific asset classes and/or securities. However, in relation to the previous research, where a backward stepwise OLS regression model was constructed as a benchmark, our results seem to conform. To the knowledge of the authors of this thesis, no research that such a model would be successful in estimating expected returns of financial time series do exist though.

In this study the estimation fails to be accurate and the model at whole produces too high portfolio turnover according to the external specifications. Hence to improve this model the portfolio optimization would have to be fed with more accurate estimations of the expected return while at the same time reducing the magnitude of

the portfolio turnover. This is possible if the estimations more frequently would be in the same direction, relative the mean of the estimations, as every actual outcome. As of now, it occurs too frequently that the estimations are on the opposite side of the mean of the estimations than the actual outcome, which adds volatility to the estimations without improving the accuracy, and in the end increases the portfolio turnover. Unfortunately, there are some limitations of the accuracy in the predictions to fulfill the external specifications, even though the correct direction would always be caught, this could, with too accurate estimations, exceed the specified restriction of the portfolio turnover. By nature the most accurate estimation would be the actual return, and intuitively, a portfolio optimization model defines optimally portfolio allocations to be where highest expected return is found, given the least risk exposure. Ideally the model would allocate according to the actual outcome of the next period. It has been found that the portfolio optimization with increased volatility of the estimations increases the magnitude of the turnover and in this study the portfolio optimization model is provided with a, by far, less volatile estimation than the volatility of the actual outcome. Whether this holds for portfolio optimization models in general will not be discussed. Conclusively, generating ANN estimations more similar to the actual outcome and provide it with either stochastic programming or mean variance as portfolio optimization model while reducing the portfolio turnover, is possible, but the accuracy is restricted by the model's context of use, in this case a predefined maximum portfolio turnover.

The objective of this thesis was to develop artificial neural networks that predict time varying expected return of financial time series with purpose of optimizing portfolio weights. In order to fulfill the objective, artificial neural networks were constructed for 15 financial time series from a broad set of asset classes. Further, a stochastic programming model for optimizing portfolio weights using the estimations was developed. The artificial neural networks estimated the time varying expected returns during 192 weeks for which the portfolio weights for seven risk levels were optimized. The results stemming from the model was not shown to outperform the benchmarks for the predictions nor the portfolio weight optimization. The indication is that the estimations from our model is highly unlikely to be better than the reference methods. Conclusively, no evidence is provided that the proposed model could predict accurately. Also, regardless of the accuracy the vast portfolio turnover of the model is not congruent with the context for which the model hypothetically would be used. Nevertheless, the objective of the thesis was fulfilled.

6.2 Discussion

The potential of predicting the return of financial time series is a widely disputed area. The topic seems to be a watershed between researchers, professional investors and private investors, and implies a fundamental difference in the approach to financial markets by the two camps. Given the weak form of the efficient market hypothesis, none of the technical data publicly available can be used to predict the future return of financial time series, and as such the relevance of artificial neural networks for financial time series prediction would be close to zero. Even if machine learning in general, and neural networks in particular, is increasingly popular in data analytics, and given at-

tention in many different domains, the fact remains; if the input to the model contains no information about future returns, the model will not succeed. The prosperity in other domains is based on the ability to find existing complex relationships between inputs and outputs, such as; the relationship between mammograms and breast cancer; the relationship between physiological measures and cardiac abnormalities. From this study and thus to the knowledge of the authors, there is no proof that sufficient relationship between historical data and future return exists for the neural network methodology to succeed in predicting financial time series.

Since this study comprised a vast number of different financial asset classes, it was problematic to determine which inputs to be used in the model. Assumingly, these financial time series do exhibit different statistical properties and dependencies, and as such their networks respectively require different informative inputs. In our model, the inputs to each asset class was adapted rather data driven with a Pearson correlation test. It would be of interest to investigate the performance of artificial neural networks using empirically determined risk premiums in the market instead, e.g. the five factors in Fama and French (2015): the market, size, book-to-market, profitability and the investment factor. Investigating the potential use of artificial neural networks to relaxe the linearity assumption of multi-factor models to improve the estimations would be of great interest. Such study would though limit the asset class that could be studied to specific equities, but a potentially successful model could be used by S&P in their discretionary portfolio management. Also, if trying to predict return for specific equities, expertise for what information the return of the specific company is dependent on could be utilized to render relevant additional explaining inputs.

Regardless of financial model there is of utmost importance that the model is robust. For which robust in this context is defined as that the estimates of the model converges to a narrow set of solutions for every run. As a professional advisor the specific advice must be deterministic not dependent of a stochastic model. One advantage of artificial neural network that has been shown to be a dilemma in this context is the fact that the functional relationship between informative data and estimations is not predefined. The functional relationship is determined through a stochastic optimization model for each financial time series to be predicted. This sources problems with the robustness. Given two people running this model with the exact same conditions, one might get a network predicting a specific outcome for the next period, and the other a network predicting the same time series but with a different outcome. On the other hand a trained network will predict the same return given the same input no matter when its run, regardless if this model is delivered as a trained networks or left to be trained by the user, this causes a huge decrease of its predictive reliability. There are two ways of overcoming this dilemma identified. The obvious way would be to just eliminate the stochastic of the results by always conducting the same random search and the same initial conditions, but the relevance of such a model is not higher since the deterministic results still would be dependent on the locked stochastic parameters. Further this would imply to remove the model selection algorithm which would by nature of optimization decrease its potential. Another way would be by utilizing grid search for infinite large set of hyper parameter, and use an infinite amount of initial conditions on the weights. A consequence of this solution is the increase of the complexity in the model, the computational time would increase tremendously. One could compare this

to CAPM where the complexity comprises to determine one parameter and keep the Ockhams' Razor[1] in mind, and the choice of model would be obvious. The optimal solution though would be to narrow the number of possible solutions, which implies narrowing the limits for the hyper-parameters. A fact is that neural networks creates a functional relationship, thus in order to reduce the limits the designer requires a deep understanding of how the parameter choices affects the functional relationship created. On top of that, the designer also needs an understanding of the function that is to be approximated, i.e. what is the pattern between the input data and the output.

Another dilemma of neural networks to be enhanced is the problem of overfitting the data. It has been identified that this is a hard nut to crack. Of course there is of interest to identify the patterns in the training data and create functional relationship that describe it as good as possible. But by the definition of overfitting, an all too good trained network will not generalize to out of sample data. This is a conflict of interest and the optimal trade off between a trained and a generalized network is hard to find. In this thesis this problem has been treated by early stopping and regularization to CAPM. Unfortunately overfitting it is still a problem in this model and whether it is right approach to use regularization is hard to say. However the optimal magnitude of the regularization coefficient have not been found and the optimal method to overcome this problem remains to be explored. This study though do add to the existing research, the methodology of regularizing the error function against CAPM. The technique of regularizing the errors against a traditional economic model do show some promising results. The prediction performance of our artificial neural networks were distinctly improved when regularizing against the CAPM prediction, where the hypothesis is that the networks became less overfitted. Also, the status quo in the literature of minimizing the quadratic errors was challenged, and a broad range of exponents of the errors was tested. However, further examination of the usefulness of other exponents than the quadratic together with CAPM regularization would be of interest. Further, it should be adviced for researchers in the area of estimating the expected return of financial time series, to evaluate their results against relevant economic models, something that far from all of the researchers in the field have done. Also, the statistical significance of the findings should be presented in order to be able to draw statistical conclusions. We propose the widely used CAPM as a benchmark model estimating equity, and the Diebold and Mariano (1995) test for testing for forecast inequality.

Further research areas

Potential future research areas proposed can be summarized as:

- Investigate multi-factors as inputs to artificial neural networks for stock prediction in order to relax linearity assumptions
- Investigate the implication of how different parameter settings limits the functional approximation, with purpose of narrowing the number of free parameters
- Further investigate the potential use of error function exponents not equal to two together with the technique of economic model regularization

[1] "Accept the simplest explanation that fits the data" - increased complexity increases the possibility of errors

6.3 Ethical aspects

An ethical aspect of this thesis is the potential misuse of similar model as the one proposed. Complex financial models that gear the investor with a distinct informational advantage compared to the market could generate excess return to the market. In order to bring transparency to the financial market, such models should be documented and made publicly available for the market to understand which factors that an investor get compensated for being exposed to, and how they get compensated. The predictive power of such model could still be valuable, even if the information is priced into the asset price, as the model then could be used to create wanted risk exposures. The findings in this thesis do however not give any investor informational advantage, and as such the necessity to make it publicly available is low.

When conducting a study to determine a model for predicting further returns, the researcher is faced with several ethical considerations, as the conclusions will be drawn by the performance on a test set. The researcher should by no means investigate the characteristics of the test data set before deigning the model, else he or she can be biased when designing the model by the gathered knowledge. By acquiring such knowledge, the researcher might unconsciously end up with a model better than he or she could do in a real setting. The authors of this thesis have maintained a strong moral compass, and evaluated the out-of-sample performance first when the model was specified. The sensitivity analysis was conducted subsequent to the actual model. As such, there is an infinitesimal risk that we were biased by the characteristics in the test set when determining the model.

Bibliography

Agatonovic-Kustrin, S. and Beresford, R. (1999). Basic concepts of artificial neural network (ann) modeling and its application in pharmaceutical research. *Journal of Pharmaceutical and Biomedical Analysis*.

Alizadeh, F., Haeberly, J.-P. A., and Overton, M. L. (1998). Primal-dual interior-point methods for semidefinite programming: Convergence rates, stability and numerical results. *SIAM Journal on Optimization*, 8(3).

Amenc, N. and Sourd, V. L. (2003). *Portfolio Theory and Performance Analysis*. Wiley.

Andersen, T. G. and Bollerslev, T. (1998). Answering the skeptics: Yes, standard volatility models do provide accurate forecasts. *International Economic Review*, 39(4).

Bengio, Y. (2012). *Practical Recommendations for Gradient-Based Training of Deep Architectures*. Springer Berlin Heidelberg.

Bergstra, J. and Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13.

Bishop, C. M. (1995). *Neural Networks for Pattern Recognition*. Clarendon Press Oxford.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer.

Black, F. and Litterman, R. (1992). Global portfolio optimization. *Financial Analysts Journal*, 30(5).

Blackrock (2015). The evolution of active investing finding big alpha in big data. *https://www.blackrock.com/institutions/en-ch/literature/whitepaper/finding-big-alpha-in-big-data-en-zz.pdf*.

Blom, G., Enger, J., Englund, G., Grandell, J., and Holst, L. (2005). *Sannolikhetsteori och statistikteori med tillämpningar*. Studentlitteratur.

Blomvall, J. (2013). Garch-poisson. *Internal document, Linköping University*.

Blomvall, J. (2016). Lecture notes in tppe33 portfolio management lecture 5, factor models - apt. *Linköping University*.

Bollerslev, T. (1986). Generalized autoregressive conditional heteroskedasticity. *Journal of Econometrics*, 31.

Box, G. E. P. and Jenkins, G. M. (1970). *Time series analysis: forecasting and control.* Holden Day.

Braun, J. and Griebel, M. (2009). On a constructive proof of kolmogorov's superposition theorem. *Constructive Approximation.*

Breiman, L. (1996). Bagging predictors. *Machine Learning*, 24.

Brock, W. A. and de Lima, P. J. F. (1995). *Nonlinear time series, complexity theory and finance.* Elsevier.

Cherubini, U., Luciano, E., and Vecchiato, W. (2004). *Copula Methods in Finance.* John Wiley Sons Ltd.

Chester, D. L. (1990). Why two hidden layers are better than one. *Proceedings of the international joint conference on neural networks*, 1.

Cheung, Y.-W., Chinn, M. D., and Pascual, A. G. (2005). Empirical exchange rate models of the seventies: Do they fit out of sample? *Journal of International Money and Finance*, 24(7).

Cont, R. (2001). Empirical properties of asset returns: stylized facts and statistical issues. *Quantitative Finance*, 1.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2:303–314.

David H. Ackley, G. E. H. and Sejnowski, T. J. (1985). A learning algorithm for boltzmann machines. *Cognitive science*, 9:147–169.

Demuth, H. and Beale, M. (2002). Neural network toolbox - for use with matlab®: User's guide.

Devroye, L. (1986). *Non-uniform Random Variate Generation.* Springer-Verlag.

Diebold, F. X. (2015). Comparing predictive accuracy, twenty years later: A personal perspective on the use and abuse of diebold-mariano tests. *Journal of Business Economic Statistics*, 23(1).

Diebold, F. X. and Mariano, R. S. (1995). Comparing predictive accuracy. *Journal of Business Economic Statistics*, 13(3).

Enke, D. and Thawornwong, S. (2005). The use of data mining and neural networks for forecasting stock market returns. *Expert Systems with Applications*, 29(4).

Fama, E. F. (1970). Capital asset prices: A theory of market equilibrium under conditions of risk. *Journal of Finance*, 25.

Fama, E. F. and French, K. R. (1970). The capital asset pricing model: Theory and evidence. *The Journal of Economic Perspectives*, 25.

Fama, E. F. and French, K. R. (2015). A five-factor asset pricing model. *Journal of Financial Economics*, 1.

Fausett, L. V. (1994). *Fundamentals of Neural Networks: Architectures, Algorithms And Applications.* Pearson.

Floreano, D. and Mattiussi, C. (2008). *Bio-Inspired Artificial Intelligence Theories, Methods, and Technologies*. MIT Press books.

Forsling, G. and Neymark, M. (2011). *Matematisk analys - En variabel*. Liber.

Freitas, F. D., de Souza, A. F., and de Almeida, A. R. (2009). Prediction-based portfolio optimization model using neural networks. *Neurocomputing*, 72(10-12).

Gangal, A. S., Kalra, P. K., and Chauhan, D. S. (2007). Performance evaluation of complex valued neural networks using various error functions. *International Journal of Electrical, Computer, Energetic, Electronic and Communication Engineering*, 1(5).

Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks.

Granger, C. W. J. and Newbold, P. (1977). *Forecasting economic time series*. Academic Press.

Grossman, S. J. and Stiglitz, J. E. (1980). On the impossibility of informationally efficient markets. *The American Economic Review*, 70(3).

Guenther, F. H. (2001). Neural networks: Biological models and applications. *International Encyclopedia of the Social & Behavioral Sciences*.

Gómez-Ramos, E. and Venegas-Martínez, F. (2013). A review of artificial neural networks: How well do they perform in forecasting time series? *Journal of Statistical Analysis*, 6(2).

Hagan, M. T., Demuth, H. B., Beale, M. H., and Jesús, O. D. (2014). *Neural Network Design second edition*. Martin T. Hagan.

Han, J., Kamber, M., and Pei, J. (2012). *Data Mining - Concepts and Techniques*. Morgan Kaufmann.

Hann, T. H. and Steurer, E. (1996). Much ado about nothing? exchange rate forecasting: Neural networks vs. linear models using monthly and weekly data. *Neurocomputing*, 10:323–339.

Hanna, S. D., Gutter, M. S., and Fan, J. X. (2001). A measure of risk tolerance based on economic theory. *Journal of Financial Counseling and Planning; Columbus*, 12.

Hansen, P. R. and Lunde, A. (2005). A forecast comparison of volatility models: Does anything beat a garch(1,1)? *Journal of Applied Econometrics*, 20.

Haykin, S. (1999). *Neural Networks - A Comprehensive Foundation. Second Edition*. Pearson Education Inc.

Haykin, S. (2009). *Neural Networks and Learning Machines*. Pearson Education Inc.

Hebb, D. O. (1949). *The Organization of Behavior*. New york: Wiley.

Hill, T., O'Connor, M., and Remus, W. (1996). Neural network models for time series forecasts. *Management Science*, 42(6).

Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79:2554–2558.

Hornik, K. (1991). Approximation capabilites of multilayer feedforward networks. *Neural Networks*, 4:251–257.

Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366.

Hsu, M.-W., Lessmanna, S., Sunga, M.-C., Maa, T., and Johnson, J. E. (2016). Bridging the divide in financial market forecasting: machine learners vs. financial economists. *Expert Systems With Applications*, 61.

Hu, M. Y., Zhang, G., Jiang, C. X., and Patuwo, E. (1999). A cross-validation analysis of neural network out-of-sample performance in exchange rate forecasting. *Decision Sciences*, 14.

Huang, W., Nakamori, Y., and Wang, S. (2004). A general approach based on autocorrelation to determine input variables of neural networks for time series forecasting. *Journal of Systems Science and Complexity*, 17.

Hull, J. C. (2009). *Options, futures, and other derivatives*. Pearson.

Hull, J. C. (2012). *Risk Management and Financial Institutions*. John Wiley Sons.

Ippolito, R. A. (1989). Efficiency with costly information: A study of mutual fund performance, 1965-1984. *The Quarterly Journal of Economics*, 104(1).

Janfalk, U. (2014). *Linjär Algebra*.

Jensen, M. C. (1967). The performance of mutual funds in the period 1945-1964. *Journal of Finance*, 23(2).

J.P.Morgan/Reuters (1996). Riskmetrics - technical document.

Kaastra, I. and Boyd, M. (1996). Designing a neural network for forecasting financial and economic times series. *Neurocomputing*, 10(3).

Kavzoglu, T. (1999). Determining optimum structure for artificial neural networks. *In Proceedings of the 25th Annual Technical Conference and Exhibition of the Remote Sensing Society*.

Kima, T. Y., Ohb, K. J., Kimc, C., and Doa, J. D. (2004). Artificial neural networks for non-stationary time series. *Neurocomputing*, 61.

LeCun, Y. A., Bottou, L., Orr, G. B., and Müller, K.-R. (1998). *Efficient Backprop*. Springer.

Lintner, J. (1965). The valuation of risk assets and the selection of risky investments in stock portfolios and capital budgets. *The Review of Economics and Statistics*, 47(1).

Lo, A. W. (2004). The adaptive markets hypothesis: Market efficiency from an evolutionary perspective. *The Journal of Portfolio Management*, 30(5).

Lo, A. W. and MacKinlay, A. C. (1988). Stock market prices do not follow random walks: Evidence from a simple specification test. *The Review of Financial Studies*, 1(1).

Lo, A. W., Mamaysky, H., and Wang, J. (2000). Foundations of technical analysis: Computational algorithms, statistical inference, and empirical implementation. *The Journal of Finance*, 55(4).

Luenberger, D. G. (1998). *Investment Science*. Oxford University Press.

Lundgren, J., Rönnqvist, M., and Värbrand, P. (2008). *Optimeringslära*. Studentlitteratur.

Maasoumi, E. and Racine, J. (2002). Entropy and predictability of stock market returns. *Journal of Econometrics*, 107.

Malkiel, B. G. (2003). The efficient market hypothesis and its critics. *Journal of Economic Perspectives*, 17(1).

Malkiel, B. J. (1973). *A random walk down Wall Street*. W. W. Norton Company, Inc.

Markowitz, H. (1952). Portfolio selection. *The Journal of Finance*, 7(1).

McCulloch, W. S. (1943). *A Logical Calculus of Ideas Immanent in Nervous Activity*. The University of Chicago Press.

Meese, R. and Rogoff, K. (1983). Empirical exchange rate models of the seventies: Do they fit out of sample? *Journal of international economics*, 3(24).

Meese, R. and Rogoff, K. (1988). Was it real? the exchange rate-interest differntial relation over the modern floating-rate period. *The Journal of Finance*, 43(4).

Mussa, M. (1986). Nominal exchange rate regimes and the behavior of real exchange rates: Evidence and implications. *Carnegie-Rochester Conference Series on Public Policy*, 25(7).

Naftaly, U., Intrator, N., and Horn, D. (1997). Optimal ensemble averaging of neural networks. *Network: Comput. Neural Syst.*, 8.

Ng, A. (2016). *Machine Learning*. Lecture Notes in Machine Learning. Coursera, Stanford University.

Nguyen, D. and Widrow, B. (1990). Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights. *1990 IJCNN International Joint Conference*.

Nielsen, M. A. (2015). *Neural Networks and Deep Learning*. Determination Press.

Perrone, M. P. (1993). Improving regression estimation: Averaging methods for variance reduction with extensions to general convex measure optimization. *PhD Thesis Brown University*.

Persson, J. and Böiers, L.-C. (2005). *Analys i flera variabler*. Studentlitteratur.

Pesaran, E. H. and Timmermann, A. (1992). A simple nonparametric test of predictive performance. *Journal of Business Economic Statistics*, 10(4).

Prechelt, L. (1998). *Early Stopping — But When?* Springer.

Qi, M. and Zhang, G. P. (2001). An investigation of model selection criteria for neural network time series forecasting. *European Journal of Operational Research*, 132.

Rahman (1968). *A Course in Theoretical Statistics*. Charles Griffin and Company.

Roll, R. (1977). A critique of the asset pricing theory's tests. *Journal of Financial Economics*, 4.

Rolls, E. T. and Treves, A. (1998). Neural networks and brain function. *Oxford university press*.

Sarle, W. S. (2002). Neural network faq, periodic posting to the usenet newsgroup comp.ai.neural-nets.

Shapiro, A., Dentcheva, D., and Ruszczynski, A. (2009). *Lectures on Stochastic Programming Modelling And Theory*. SIAM.

Sharpe, W. F. (1964). Efficient capital markets: A review of theory and empirical work. *Journal of Finance*, 19(3).

Sharpe, W. F. (1966). Mutual fund performance. *The Journal of Business*, 39(1).

Siegelmann, H. T., Horne, B. G., and Giles, C. L. (1997). Computational capabilities of recurrent narx neural networks. *IEEE Transactions on Systems, Man, and Cybernetics*, 27:208–215.

Siegelmann, H. T. and Sontag, E. D. (1991). Turing computability with neural nets. *Applied Mathematics Letters*, 4:77–80.

Sklar, A. (1959). Fonctions de répartition à n dimensions et leurs marges. *Publ. Inst. Statist. Univ. Paris*.

Szeliga, M. I., Verdes, P. F., Granitto, P. M., and Ceccatto, H. A. (2003). Artificial neural network learning of nonstationary behavior in time series. *International Journal of Neural Systems*, 13(2).

Timmermann, A. and Granger, C. W. (2004). Efficient market hypothesis and forecasting. *International Journal of Forecasting*, 20.

Treynor, J. L. (1965). How to rate management of investment funds. *Harvard Business Review*, 43(1).

Trivedi, P. K. and Zimmer, D. M. (2005). Copula modeling: An introduction for practitioners. *Foundations and Trends in Econometrics*, 1.

Tsai, C.-F. and Hsiao, Y.-C. (2003). Combining multiple feature selection methods for stock prediction: Union, intersection, and multi-intersection approaches. *Decision Support Systems*, 50.

Walczak (2001). An empirical analysis of data requirements for financial forecasting with neural networks. *Journal of Management Information Systems*.

Widrow, B., Rumelhart, D. E., and Lehr, M. A. (1997). Neural networks: Applications in industry, business and science. *Communications of the ACM*, 37(3).

Zhang, G., Patuwo, E., and Hu, M. Y. (1998). Forecasting with artificial neural networks: The state of the art. *International Journal of Forecasting*, 14.

Zilouchian, A. (2001). *Fundamentals of Neural Networks*. CRC Press.

Appendix A

Variable declaration

	One-dimensional	Vector	Matrix
Generic variable	x	$\mathbf{x}$	$\mathbf{X}$
Stochastic variable	X	$\underline{\mathrm{X}}$	$\underline{\underline{\mathrm{X}}}$

This holds except for all capital letters declared below

Variables

Generic

Notation	Description
x	Function input
u	Uniformly distributed variable as $\sim U[0,1]$
y	Function output
r	Time series return
μ	Expected value
σ	Volatility
σ^2	Variance
Cov	Covariance
ρ	Correlation
ϱ	Autocorrelation
q	Component of eigenvector
κ	Principal component
ϵ	Error
ε	Independently identically distributed error
p	probability
S	Price of general time series
S_r	Sharpe ratio

Phase 1 specific

x	Input
h	Hidden node
a	Activation of random node
ω	Network weights
z	Output from random node
y	Output from network
μ_n	Expected return Neural Network
μ_b	Expected return benchmark model
w	Network weights

Phase 2 specific

x^{buy}	Buy decision
x^{sell}	Sell decision
w	Portfolio weights
W	Wealth
R_n	Portfolio return Neural Network
R_b	Portfolio return benchmark portfolio
ξ	help-variable stochastic programming

Parameters

Generic

Notation	Description
θ	Random unknown parameter
n, k	Random discrete number
t	Random time point
T	Random set of time point
α, β, ι	Random constants
λ	Decay factor EWMA, Eigenvalue, hyper parameter
$\boldsymbol{\beta}$	Systematic risk
p,q	GARCH parameter

Phase 1 specific

L	Number of Layers
M	Number of nodes
γ	Risk aversion
η	Learning rate
ϕ	Regularization term
K	Crossvalidation folds
N	Number of networks in validation set
N_t	Number of training examples
N_v	Number of validation examples
N_{te}	Number of testing examples
τ	Lag periods

Phase 2 specific

I	Number of assets
S	Number of Monte Carlo simulated scenarios
$i \in \{1, \ldots, I\}$	Number i of the asset x_i in the portfolio universe
$s \in \{1, \ldots, S\}$	Number s of the Monte Carlo generated scenarios
Δt	Monte Carlo simulation time period
x_i^{init}	Initial holding in asset i
$r_i^{(s)}$	Return for asset i in scenario s in SEK
c^{init}	Initial cash
r_f	Return for the riskfree asset in SEK
p_s	Probability for scenario s occurring
γ	Risk aversion parameter
τ	Transaction cost

Functions and models

Generic

Notation	Description
$g(\cdot)$, $f(\cdot)$	Random function
$\Pr(\cdot)$	Probability
$\text{pdf}(\cdot)$	General probability density function
$\text{cdf}(\cdot)$	General cumulative distribution function
$F(\cdot)$	General cumulative distribution function
$E[\cdot]$	Expected value
$\text{Var}[\cdot]$	Variance
$\text{Cov}\ [\cdot]$	Covariance
$c(\cdot)$	Copula probability density function
$C(\cdot)$	Copula cumulative distribution function
$\Phi(\cdot)$	Student's t distributed copula
$\text{Li}(\cdot)$	Likelihood function

Model specific

$\varphi(\cdot)$	Activation function
$\varphi_h(\cdot)$	Activation function hidden neurons
$\varphi_o(\cdot)$	Activation function output neurons
$U(\cdot)$	Utility function
$J(\cdot)$	Error function

Index

Generic

Notation	Description
i, j	Random index
τ	Random time lag index

Model specific

l	Layer index

τ

Appendix B

Data description

Table B.0.1: Assets classes used for forecasting and portfolio optimization. The FX-rates were not used in the portfolio optimization. The first column is an ID for the asset class. The second column describes the asset class. The third column describes the index name in Bloomberg. The fourth column describes the unique identifier in Bloomberg. The fifth column contains the firt data point of the time series, and the sixth column contains the length of the time series sampled at weekly frequency.

No.	Assets	Index	BBG Ticker	Start	Data points
1	OMX Index	Swedish equity	OMX Stockholm 30 Index	1986-12-19	1586
2	MXWO Index	Global equity	MSCI AC World Total Return Index	1971-01-08	2404
3	RXBO Index	Interest rate	OMRX Total Bond Index	1996-09-27	1062
4	RXRE Index	Inflation	OMRX Real Return Index	1996-10-18	1059
5	RXVX Index	Cash	OMRX Treasury Bill Index	1996-09-27	1062
6	HE00 Index	Credit	Merrill Lynch Euro High Yield Constrained	1998-01-02	1010
7	LEGATRUU Index	Global bond	Bloomberg Barclays Global-Agg. Tot. Return Index	1990-02-02	1423
8	JGENBDUU Index	EM currency/bond	JPMorgan GBI EM Broad Diversified Index	2003-01-03	735
9	MXWO0RE Index	Real estate	MSCI World Real Estate Index	1994-12-30	1167
10	BCOMTR Index	Commodity	Bloomberg Commodity Index Total Return	1991-01-04	1375
11	HFRXM Index	CTA	HFRX Macro/CTA Index	1998-01-30	992
12	HFRXAR Index	Absolute return	HFRX Absolute Return Index	1998-01-02	996
13	JPYSEK Index	JPYSEK	-	1971-01-08	2404
14	EURSEK Index	EURSEK	-	1998-12-30	1466
15	USDSEK Index	USDSEK	-	1971-01-08	2404

B.1 Macro factors

Table B.1.1 presents the macro factors used as potential inputs. The macro factors are only used when they are available, i.e. GDP for Q1 is used first in Q2 etc. Abbreviations used in the table:

- YoY - Year on Year, meaning the percentual change in relation to the same period last year. Appropriate for data that shows seasonality.
- QoQ - Quarter on Quarter, meaning the percentual change in relation to the last quarter. Appropriate for data that do not show seasonality.
- SA - Seasonally Adjusted data.

- NSA - Not Seasonally Adjusted data.

Short name	Index	First	Frequency	Comment
PurchasingManagerIndex	NAPMPMI	1960	Monthly	Nomin. Manufac.
USProduction	IP YOY	1960	Monthly	YoY%
JPYIndustrialProduction	JNIPYOY	1979	Monthly	YoY%
ChinaProduction	CHVAIOY	1990	Monthly	YoY%
EURProduction	EUITEMUM	1991	Monthly	YoY%
Export	MWT PEWO	2000	Monthly	Nomin. World
USConsumerConfidence	CONCCONF	1977	Monthly	Nominal SA
USGDP	GDP CUAQ	1960	Quarterly	QoQ% SA
EURGDP	EUGDEMU	1995	Quarterly	QoQ% SA[1]
SWEGDP	SWGCGDPY	1981	Quarterly	YoY% NSA
CHGDP	CNNGPQ$	1992	Quarterly	YoY% NSA[2]
JPYGDP	JGDOQOQ	1980	Quarterly	QoQ% SA
USUnemployment	USURTOT	1960	Monthly	% SA
EURUnemployment	UMRTEMU	1998	Monthly	% SA
CHUnemployment	CNUERATE	2002	Quarterly	% NSA
JPYUnemployment	JNUNRT	2001	Monthly	% NSA
USMoneySupply	M1 Index	1981	Weekly	YoY% [3]
CHMoneySupply	CNMSM1	1990	Monthly	YoY% [4]
EURMoneysupply	ECMAM1YY	1981	Monthly	YoY% [5]
JPYMoneySupply	JMNSM1	2003	Monthly	YoY% [6]
USInflation	CPI YOY	1960	Monthly	YoY%
JPYInflation	JNCPIYOY	1971	Monthly	YoY%
EURInflation	ECCPEMUY	1997	Monthly	YoY%
CHInflation	CNCPIYOY	1990	Monthly	YoY%
BalticDryIndex	BDIY	1985	Weekly	Nominal
SWEKonjunkturbarometern	SWETSURV	1996	Monthly	Nominal
PEMSCIWorld	-[7]	1995	Quarterly	Nominal
US10YYields	USGG10YR	1967	Weekly	Nominal
US3YYields	USGG10YR	1967	Weekly	Nominal
US3MYields	USGB090Y	1997	Weekly	Nominal
JPY10YYields	GJGB10	1987	Weekly	Nominal
JPY3YYields	GJGB3	1989	Weekly	Nominal

[1]Originally nominal, seasonally adjusted. Converted to QoQ as the index was seasonally adjusted

[2]Originally nominal, not seasonally adjusted. Converted to YoY as the index was not seasonally adjusted.

[3]Originally nominal, not seasonally adjusted. Converted to YoY as the index was not seasonally adjusted.

[4]Originally nominal, not seasonally adjusted. Converted to YoY as the index was not seasonally adjusted.

[5]Originally nominal, not seasonally adjusted. Converted to YoY as the index was not seasonally adjusted.

[6]Originally nominal, not seasonally adjusted. Converted to YoY as the index was not seasonally adjusted.

[7]This is not a specific index, rather a Bloomberg calculated metric

JPY3MYields	JY0003M	1986	Weekly	Nominal
EUR10YYields	GECU10YR	1993	Weekly	Nominal
EUR3YYields	GECU3YR	1994	Weekly	Nominal
EUR3MYields	EUR003M	1999	Weekly	Nominal
CN10YYields	GCNY10YR	2005	Weekly	Nominal
CN3YYields	GCNY3YR	2005	Weekly	Nominal
CN3MYields	IBO03M	1996	Weekly	Nominal

Appendix C

Estimations of volatility and correlation

Below theory for estimation of volatility and correlation is provided.

C.1 Expected volatility

The mathematical definition of the variance of a random variable X is $E[(X-\mu)^2] = \sigma^2$. In finance, volatility is a common word to refer to the standard deviation σ of the return for a financial instrument, i.e. a random variable. Standard deviation and thus the volatility is the square root of the variance.

Let us define σ_t as the volatility at time t estimated at time $t-1$. The volatility of financial time series is commonly measured on a daily basis. A traditional method for estimating the volatility for the stochastic return R of an asset is by using its historical standard deviation, using T past returns, defined as

$$\hat{\sigma}_t^2 = \frac{1}{T}\sum_{\tau=1}^{T}(r_{t-\tau=} - \bar{r})^2 \tag{C.1.1}$$

also referred to as the Simple Moving Average (SMA) model by J.P.Morgan/Reuters (1996). SMA gives equal weights to all deviations, no matter when in time they happened. Also, if one uses a moving window technique, the metric will change rapidly when extreme observations fall out of the sample window. Hull (2012); J.P.Morgan/Reuters (1996); Cont (2001) note that the volatility of financial time series is non-constant which means that there exist periods with higher and respectively lower volatility motivating introduction of a model that incorporate volatility clustering (see Section 3.1.2) which is a non-stationary attribute.

An alternative to the SMA is the Exponentially Weighted Moving Average (EWMA) model that introduces a decay factor $\lambda \in (0, 1)$ that gives recent observations higher weights. The model is, depending on decay factor choice, more responsive to recent changes, and deals with large changes associated with samples falling out of the sample window as the weights associated with the sample decreases exponentially.

J.P.Morgan/Reuters (1996) uses EWMA for volatility calculations based on the motivation that the metric provide an optimal balance given the context that most practitioners in financial risk work within, although not detailing the motivation further. An interpretation is that they find the metric good enough for most purposes. The volatility estimation is in EWMA calculated as

$$\hat{\sigma}_t^2 = (1-\lambda)\sum_{\tau=1}^{T}\lambda^{\tau=1}(r_{t-\tau}-\bar{r})^2. \tag{C.1.2}$$

based on the approximation that unity holds, i.e. the sum of the weights is equal to 1, meaning that

$$\sum_{\tau=1}^{T}\lambda^{\tau-1} \approx \frac{1}{1-\lambda}. \tag{C.1.3}$$

As $T \to \infty$ the two expressions in (C.1.3) are equivalent. Considering daily observations of the return, it holds approximately that the arithmetic mean of the returns $\bar{r} = 0$ (holds approximately for weekly returns as well), an approximation often made in volatility models (J.P.Morgan/Reuters, 1996), which reduces the model to

$$\hat{\sigma}_t^2 = (1-\lambda)\sum_{\tau=1}^{T}\lambda^{\tau-1}r_{t-\tau}^2. \tag{C.1.4}$$

The relationship can also be written in recursive form

$$\begin{aligned}\hat{\sigma}_t^2 &= (1-\lambda)\sum_{\tau=1}^{\infty}\lambda^{\tau-1}r_{t-\tau}^2 = (1-\lambda)(r_{t-1}^2+\lambda r_{t-2}^2+\lambda^2 r_{t-3}^2+\ldots)\\ &= \lambda\underbrace{(1-\lambda)(r_{t-2}^2+\lambda r_{t-3}^2+\lambda^2 r_{t-4}^2+\ldots)}_{\hat{\sigma}_{t-1}^2}+(1-\lambda)r_{t-1}^2\\ &= \lambda\hat{\sigma}_{t-1}^2+(1-\lambda)r_{t-1}^2\end{aligned} \tag{C.1.5}$$

which for computational purposes is compelling as only the latest volatility estimate and the latest return has to be available. By calculating the weighted average of individual optimal decay factors, estimated by MLE, J.P.Morgan/Reuters (1996) found that an appropriate decay factor for daily returns is 0.94. $and formonthlyreturnsis 0.97$.

Several other models exist, such as extreme value techniques, two step regression analysis, stochastic volatility, applications of chaotic s and GARCH. The Generalized Autoregressive Conditional Heteroskedasticity (GARCH) is a model that has received high attention by academia and financial companies (J.P.Morgan/Reuters, 1996). Bollerslev (1986) describes the GARCH model that accomodates mean reversion, not provided by EWMA, and leptokurtosis (heavy tails). The general GARCH(p,q) model uses p past returns and q past volatilities for estimating the volatility and is defined as

$$\hat{\sigma}_t^2 = \omega + \sum_{i=1}^{p} \alpha_i r_{t-i}^2 + \sum_{i=1}^{p} \beta_i \hat{\sigma}_{t-i}^2 \tag{C.1.6}$$

In practice, GARCH(1,1) is by far the most popular of the GARCH models (Hull, 2012), and simplifies the volatility forecast to the recursive form

$$\hat{\sigma}_t^2 = \iota V_L + \alpha r_{t-1}^2 + \beta \hat{\sigma}_{t-1}^2, \quad \iota + \alpha + \beta = 1, \quad \iota, \alpha, \beta \geq 0 \tag{C.1.7}$$

where V_L is the long run average. The EWMA is a special case of the GARCH(1,1) with $\iota = 0$, $\alpha = 1 - \lambda$ and $\beta = \lambda$ (Hull, 2012). The model can also be rewritten with $\omega = \iota V_L$ for parameter estimation purposes, which gives

$$\hat{\sigma}_t^2 = \omega + \alpha r_{t-1}^2 + \beta \hat{\sigma}_{t-1}^2, \quad \iota = 1 - \alpha - \beta, \quad V_L = \frac{\omega}{\iota}, \quad \alpha + \beta \leq 1. \tag{C.1.8}$$

Several extensions of the GARCH model have been proposed, e.g. Exponential GARCH, Integrated GARCH, Switching Regime ARCH (J.P.Morgan/Reuters, 1996), but despite its relative simple relationship, the GARCH(1,1) is found to provide relatively good predictive performance (Andersen and Bollerslev, 1998). In a study Hansen and Lunde (2005) compare the predictive performance of GARCH(1,1) with 330 more sophisticated ARCH models and find that no model outperforms GARCH(1,1) on predicting volatility on DM/USD spot rate and IBM return.

Estimation of EWMA and GARCH parameters can be done with e.g. maximum likelihood-estimation.

The calculations of volatility this far has been for one day ahead, for which the assumption of $\bar{r} = 0$ approximately holds. However, longer forecast horizons might be of interest. We have that $\iota = 1 - \alpha - \beta$ which in (C.1.7) gives

$$\hat{\sigma}_t^2 - V_L = \alpha(r_{t-1}^2 - V_L) + \beta(\hat{\sigma}_{t-1}^2 - V_L) \tag{C.1.9}$$

which is equivalent with

$$\hat{\sigma}_{t+\tau}^2 - V_L = \alpha(r_{t+\tau-1}^2 - V_L) + \beta(\hat{\sigma}_{t+\tau-1}^2 - V_L). \tag{C.1.10}$$

If we assume that $E[r_i] = 0, \forall i$ we have that $E[r_{t+\tau-1}^2] = \sigma_{t+\tau-1}^2$ which in (C.1.10) gives

$$E[\sigma_{t+t}^2 - V_L] = (\alpha + \beta) E[\sigma_{t+\tau-1}^2 - V_L]. \tag{C.1.11}$$

By repeatedly substituting $E[\sigma_{t+\tau-1}^2 - V_L]$ and noting that $E[V_L] = V_L$ we end up with

$$E[\sigma_{t+t}^2] = V_L + (\alpha + \beta)^t (\hat{\sigma}_t^2 - V_L) \tag{C.1.12}$$

which is the $\tau-$step ahead forecast of the volatility. (Bollerslev, 1986) For EWMA, $\alpha + \beta = 1$, hence

$$E[\sigma_{t+t}^2] = \hat{\sigma}_t^2. \tag{C.1.13}$$

J.P.Morgan/Reuters (1996) show that the time-scaled volatility is simply the square root of multiples of the single day forecasts meaning that

$$\hat{\sigma}_{t,T} = \sqrt{T}\hat{\sigma}_t. \tag{C.1.14}$$

To aggregate the volatility of a portfolio that consists of a set of assets, we must introduce the concept of covariance and correlation described in Section C.2.

C.2 Expected correlation

The correlation of two random variables describes to which degree the random variables have some dependence or relationship to each other. The most commonly used correlation is the Pearson correlation coefficient, which between two random variables, X and Y, is defined as

$$\rho_{X,Y} = \frac{E[(X - \mu_X)]E[(Y - \mu_Y)]}{\sigma_X \sigma_Y} = \frac{Cov[X,Y]}{\sigma_X \sigma_Y}, \quad \rho_{X,Y} \in [-1,1] \tag{C.2.1}$$

We will in this thesis define correlation as the Pearson correlation coefficient. Other correlation metrics that exist is Spearman's rank correlation coefficient and Kendall's rank correlation coefficient, but the description of these will be left out in this thesis.

The equally weighted historic correlation can be used to estimate future correlation, and is calculated from the observations as

$$\hat{\rho}_{X,Y} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2 \sum_{i=1}^{n}(y_i - \bar{y})^2}}. \tag{C.2.2}$$

Further, the covariance between X and Y, $Cov[X,Y]$, involving both variance and correlation to describe the joint variability, is obtained from equation C.2.1 as

$$Cov[X,Y] = \rho_{X,Y}\sigma_X\sigma_Y \tag{C.2.3}$$

Analogous with suggested models for estimating the volatility of an asset, a higher weight to recent observation is compelling. Therefore multivariate EWMA and multivariate GARCH models are developed to estimate the future covariance (Amenc and Sourd, 2003). The multivariate EWMA model for estimating the covariance is defined analogous to the univariate EWMA model as

$$\hat{Cov}[X,Y]_t = \lambda \hat{Cov}[X,Y]_{t-1} + (1-\lambda) r_{x,t-1} r_{y,t-1}, \quad \lambda \in (0,1). \tag{C.2.4}$$

Intuitively, the multivariate GARCH(1,1) model is hence defined as

$$\hat{Cov}[X,Y]_t = \iota V_L + \alpha r_{x,t-1} r_{y,t-1} + \beta \hat{Cov}[X,Y]_{t-1}, \quad \iota + \alpha + \beta = 1, \quad \iota, \alpha, \beta \geq 0 \tag{C.2.5}$$

J.P.Morgan/Reuters (1996) shows that time-scaling of covariances is equivalent to time-scaling of variances, which from (C.1.14) imply that

$$\hat{Cov}[X,Y]_{t,T} = \sqrt{T}\hat{Cov}[X,Y]_t \tag{C.2.6}$$

thus the correlation estimation remain unchanged regardless of time-scaling.

The covariance matrix $\mathbf{Cov} \in \mathbb{R}^{nxn}$ for n assets is of interest and defined as

$$\mathbf{Cov} = \begin{pmatrix} Cov[X_1, X_1] & \cdots & Cov[X_1, X_n] \\ \vdots & \ddots & \vdots \\ Cov[X_n, X_1] & \cdots & Cov[X_n, X_n] \end{pmatrix}. \tag{C.2.7}$$

To derive the correlation matrix, we have from (C.2.3) that each position (i, j) in $\mathbf{C}$ is simply divided by $\sigma_i \sigma_j$ determined by the univariate volatility estimating scheme for i and j respectively.

Hull (2012) specify the condition for a covariance matrix to be internally consistent as

$$\mathbf{w}^T \mathbf{Covw} \geq 0. \tag{C.2.8}$$

where $\mathbf{w} \in \mathbb{R}^{n \times 1}$ describe weights of portfolio constituents, and as such (C.2.8) corresponds to the portfolio variance, σ_p^2. The condition is intuitive from the definition of variance, i.e. the variance of a portfolio's returns has to be non-negative. A matrix that satisfies (C.2.8) is called a positive semidefinite matrix.

Appendix D

Univariate distributions

A common assumption is that financial time series follows a Geometric Brownian Motion (GBM), where the assumption is made that the returns are normally distributed. As described in Section 3.1.2 this is not necessarily the case for financial time series as these exhibit heavy tails. However, remember also the property of aggregational gaussanity, which means that when the time period for which a return is calculated is increased, the distribution of the returns are increasingly normally distributed (Cont, 2001). A student's t process is a similiar process that contrary to GBM accomodates heavier tails, a property that is shared with the GARCH-Poisson process. Below, these three processes will be described.

D.1 Geometric Brownian Motion (GBM)

Hull (2009) describes that the continuous price change, dS, of an asset with price S during the time dt is often assumed to follow a GBM such as

$$dS = \mu S dt + \sigma S dz \tag{D.1.1}$$

where μ is the expected return, σ is the expected volatility, $dz = \sqrt{dt}\epsilon$ is a Wiener process, i.e. the increments are independent and normally distributed $\epsilon \sim N(0,1)$ with a continuous path that starts at 0. A time discretized version of the percentual return using (D.1.1) is defined by

$$\frac{\Delta S}{S} = \mu \Delta t + \sigma \sqrt{\Delta t}\epsilon \tag{D.1.2}$$

where $\Delta S = S_{\Delta t} - S$ is the absolute price change between the price, after time increment Δt, $S_{\Delta t}$, and S.The asset price trajectory can be simulated during the period $N\Delta t$ by repeatedly drawing random variates from $\epsilon \sim N(0,1)$ N times.

In order to model a variable that is log-normally distributed (i.e. a normal distribution of the log-return, see Section 3.3) one can utilize Itô's lemma (Hull, 2009) to receive the process

$$d\ln S = \left(\mu - \frac{\sigma^2}{2}\right)dt + \sigma dz \tag{D.1.3}$$

which is discretized to

$$\ln\left(\frac{S_{\Delta t}}{S}\right) = \left(\mu - \frac{\sigma^2}{2}\right)\Delta t + \sigma\sqrt{\Delta t}\epsilon. \tag{D.1.4}$$

where $S_{\Delta t}$ is the price after the time increment Δt. Generally, by assuming that the GBM holds, one assumes that an underlying stochastic variable is normally distributed $X \sim N(\mu, \sigma)$ with probability density function (PDF)

$$f_X(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}. \tag{D.1.5}$$

(Blom et al., 2005)

D.2 Student's t distribution

The student's t distribution can deal with heavy tails, which is a characteristic of many financial times series (see Section 3.1.2). The student's t distribution has an extra parameter which is the degrees of freedom v. When $v \to \infty$ the student's t distribution converges to the normal distribution. By letting the stochastics in the GBM be explained by a random variate $\epsilon \sim t(v, 0, 1)$ one can model underlying stochastic variables that are student's t distributed. The PDF for the student's t distributed variable X is given by

$$f_X(x) = \frac{\Gamma(\frac{v+1}{2})}{\sqrt{n\pi}\Gamma(\frac{v}{2})(1+\frac{x^2}{v})^{\frac{(v+1)}{2}}}, \quad \Gamma(t) = \int_0^\infty x^{t-1}e^{-x}dx \tag{D.2.1}$$

(Blom et al., 2005)

D.3 GARCH-Poisson process

Blomvall (2013) introduces the GARCH-Poisson process which accomodates heavy tails as well as models sparse jumps in the asset returns. The GARCH-Poisson process for log-returns with time step Δt is given by

$$\begin{aligned} \ln\left(\frac{S_{i+1}}{S_i}\right) &= \mu_i\Delta t + \sigma_i\epsilon\sqrt{\Delta t} + \partial\sigma_i\sqrt{h}\xi\sqrt{\Delta t} \\ \sigma_{i+1}^2 &= \beta_0 + \beta_1\sigma_i^2 + \beta_2\frac{(\ln(\frac{S_{i+1}}{S_i}) - \gamma\Delta t)^2}{\Delta t} \\ \epsilon \sim N(0,1), &\quad \xi \sim N(0,1), \quad h \sim Po(\lambda\Delta t) \end{aligned} \tag{D.3.1}$$

where ∂ and λ are parameters, β_i, $i = 0, 1, 2$ are weights for updating the volatility. The ϵ term models the effect of continuous information flow to the the market on the asset return. However, some news are less frequent, and creates jumps in the asset returns, e.g. company specific information. Such information is modelled by ξ which is scaled by $\partial\sqrt{h}$.

The probability density function $f_{Xi}(x_i)$ for a stochastic variable X_i that follows a GARCH-Poisson process is given by

$$f_{Xi}(x_i) = \frac{1}{\sqrt{2\pi\sigma_i^2\Delta t}} e^{-\lambda\Delta t} \sum_{k=0}^{\infty} \frac{1}{\sqrt{1+\partial^2 k}} e^{-\frac{1}{2}\frac{(x_i-\mu_i\Delta t)^2}{\sigma_i^2(1+\partial^2 k)\Delta t}} \frac{(\lambda\Delta t)^k}{k!} \tag{D.3.2}$$

The parameters in the probability function can be estimated using MLE. In order to be able to use QQ-plots and Monte Carlo simulation for the GARCH-Poisson process, the cumulative distribution function is needed to invert random variates and is given by

$$F_{Xi}(x_i) = \sum_{k=0}^{\infty} N\left(\frac{x_i - \mu_i\Delta t}{\sigma_i\sqrt{(1+\partial^2 l)\Delta t}}\right) \frac{(\lambda\Delta t)^k}{k!} e^{-\lambda\Delta t} \tag{D.3.3}$$

where $N(\cdot)$ is the cumulative distribution function for a standard normal distribution (i.e. expected value 0 and variance 1).

Appendix E

Plots for different error functions

E.1 Plots for different exponents

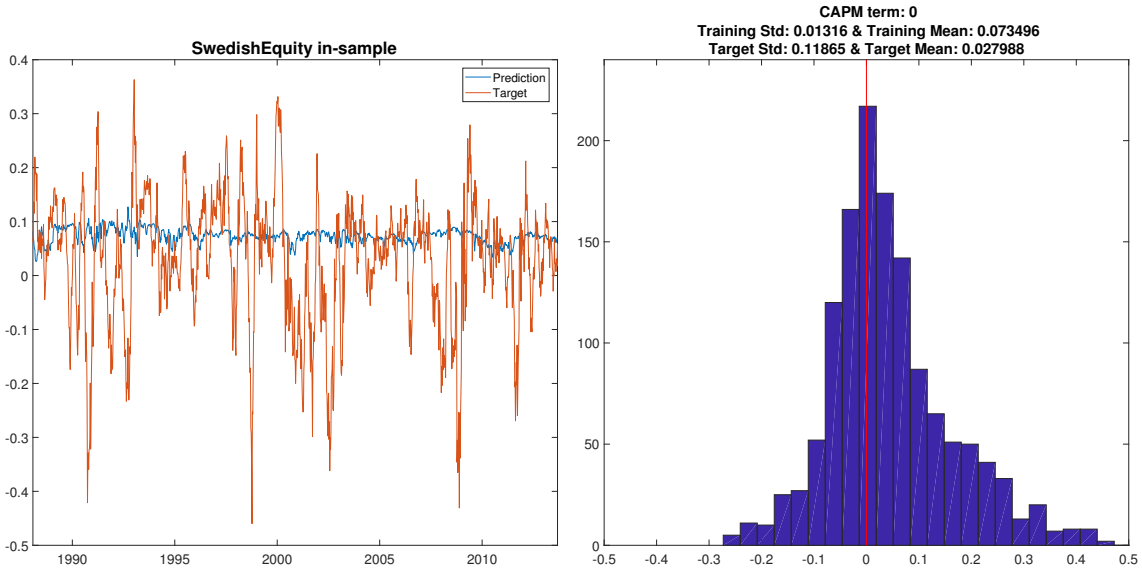

Figure E.1.1: Prediction plot and error histogram of error function $J = |y - \hat{y}|^{0.1}$

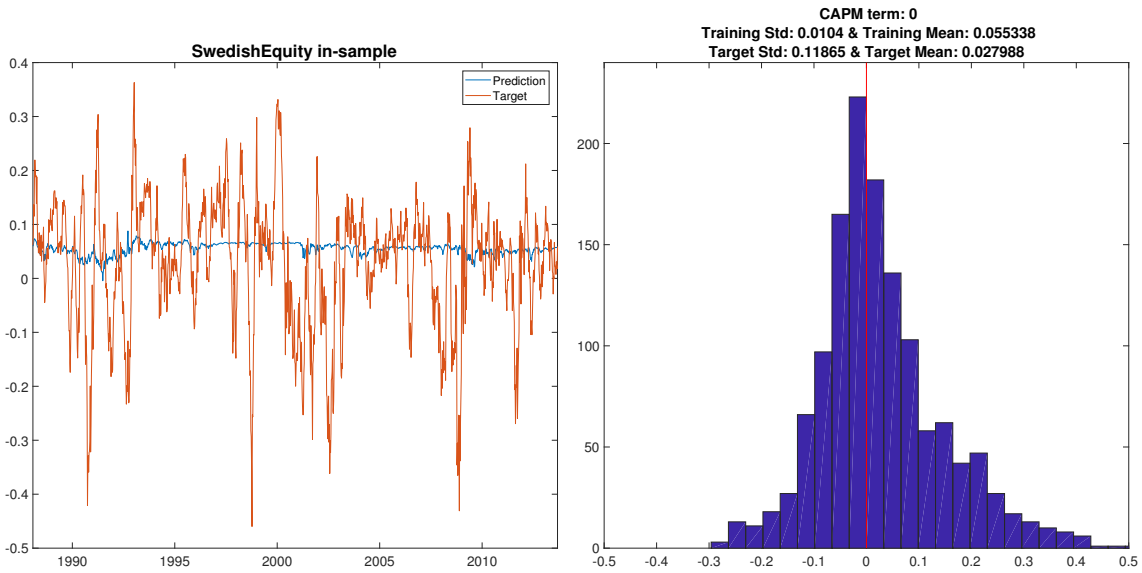

Figure E.1.2: Prediction plot and error histogram of error function $J = |y - \hat{y}|^{0.5}$

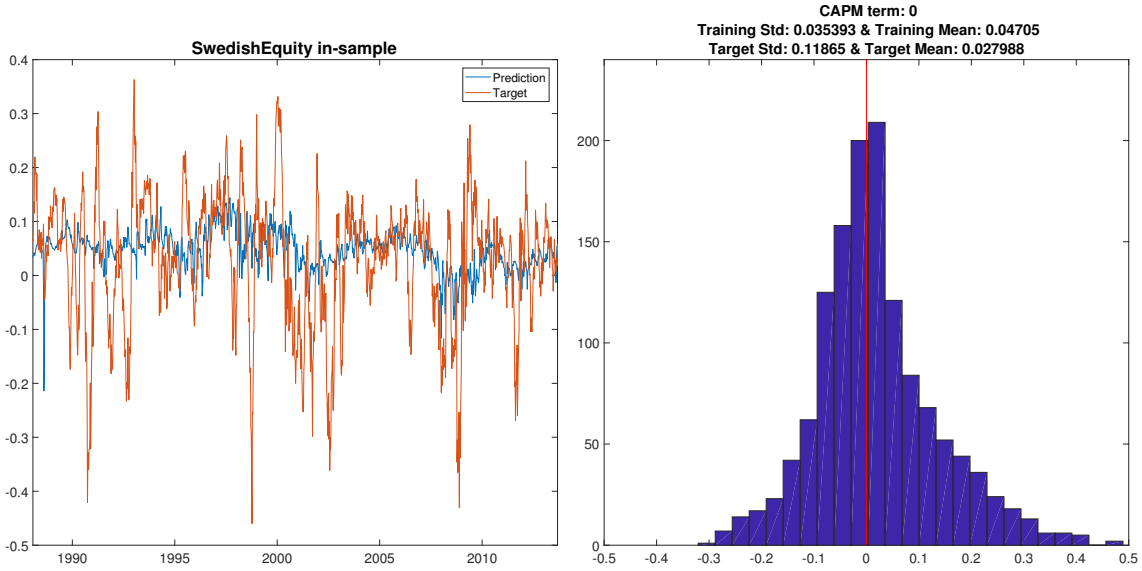

Figure E.1.3: Prediction plot and error histogram of error function $J = |y - \hat{y}|$

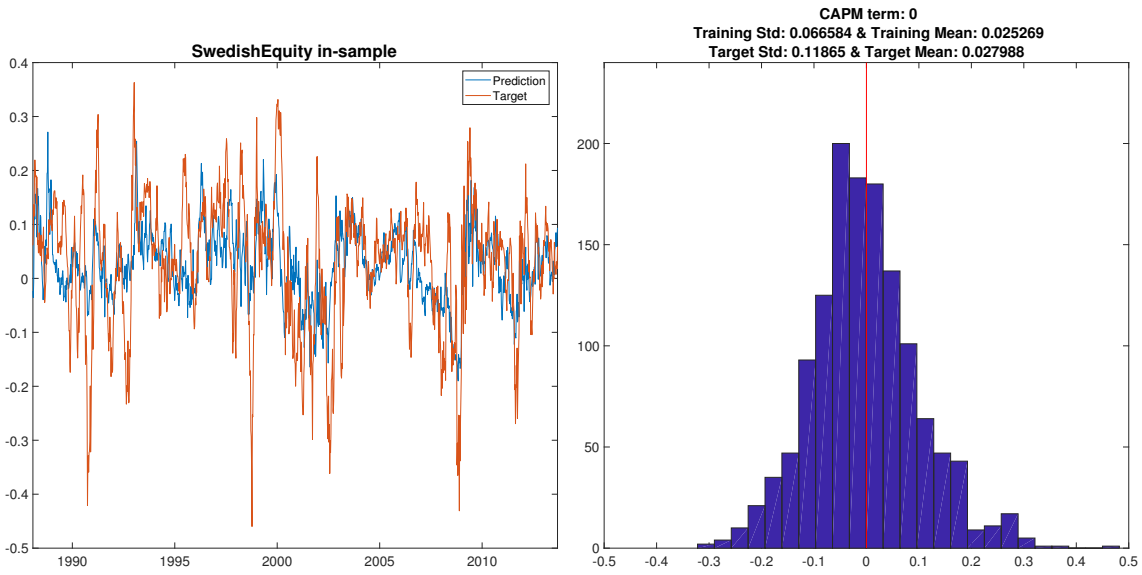

Figure E.1.4: Prediction plot and error histogram of error function $J = |y - \hat{y}|^2$

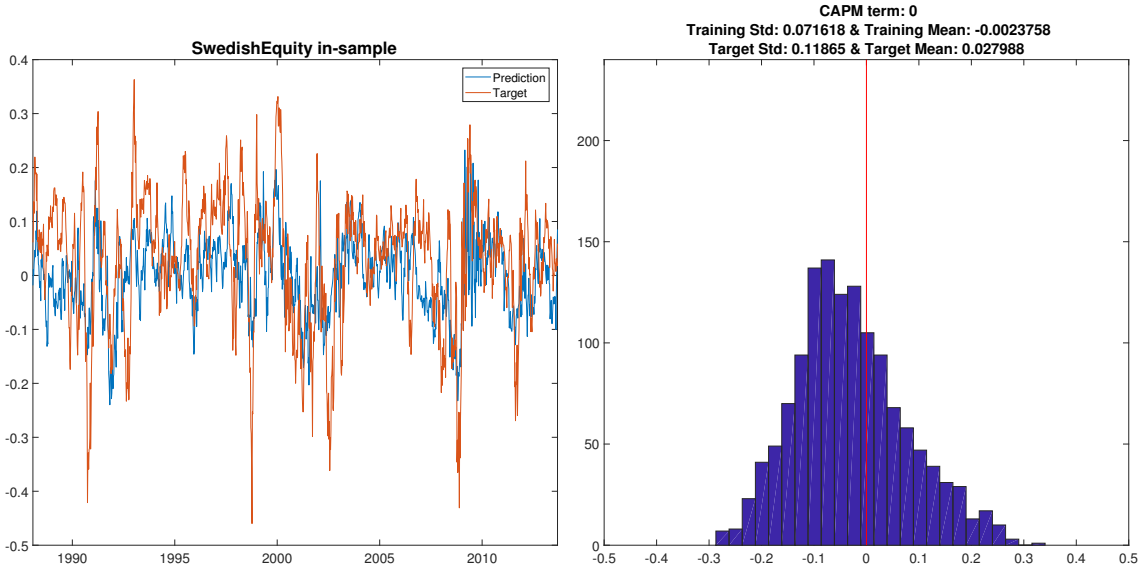

Figure E.1.5: Prediction plot and error histogram of error function $J = |y - \hat{y}|^6$

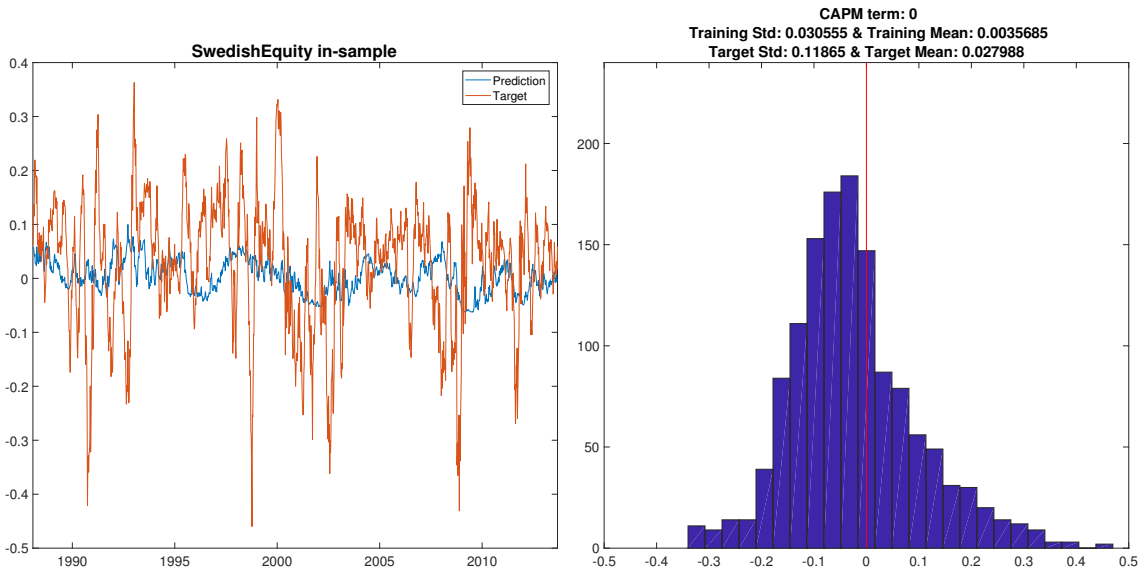

Figure E.1.6: Prediction plot and error histogram of error function $J = |y - \hat{y}|^{26}$

E.2 Plots for different CAPM regularization coefficient

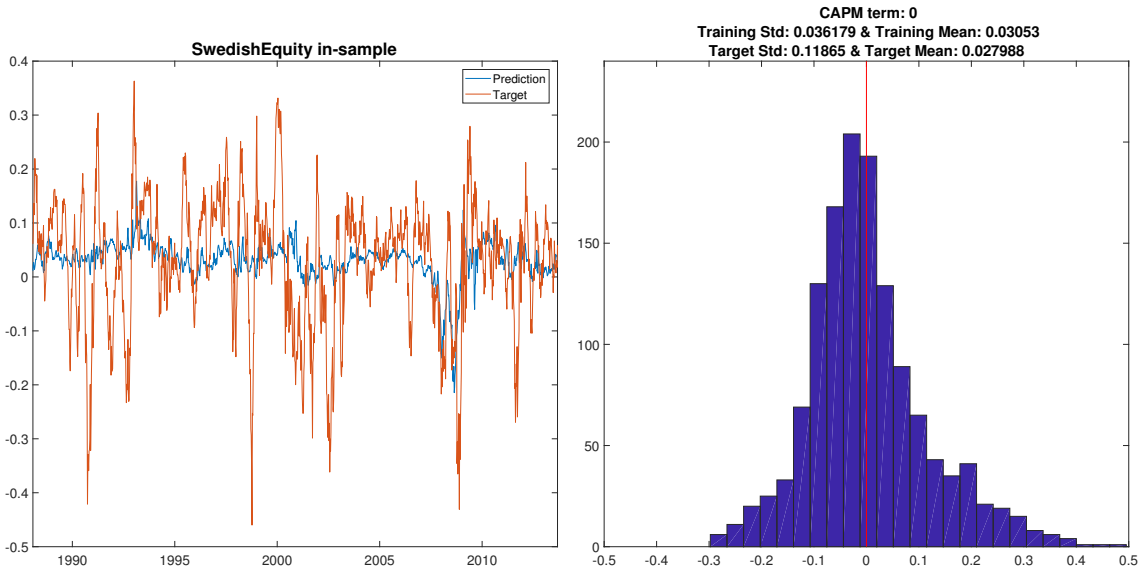

Figure E.2.1: Prediction plot and error histogram of error function $J = |y - \hat{y}|^2$

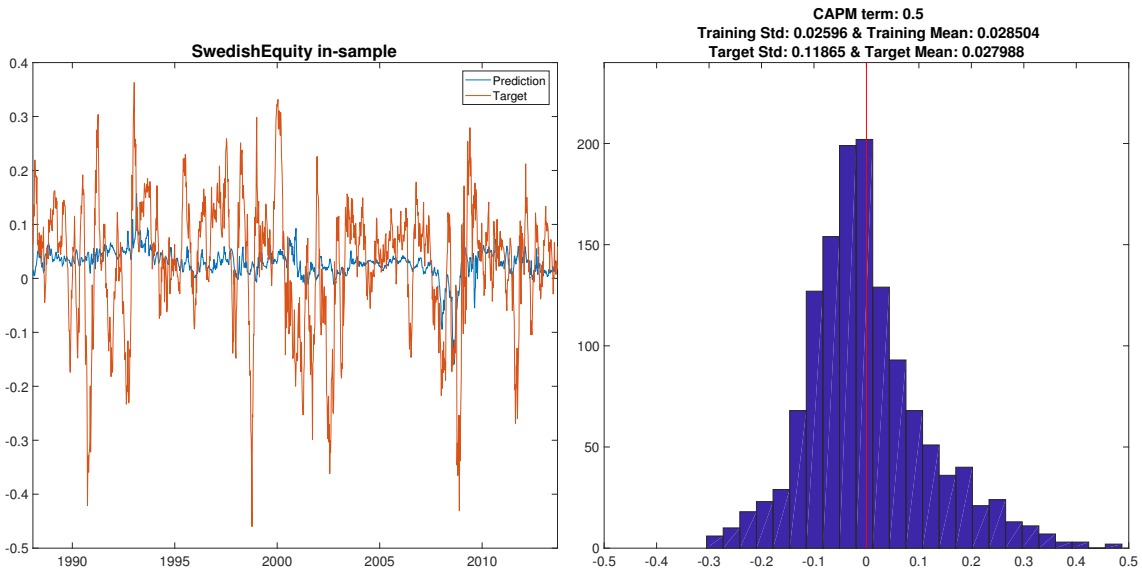

Figure E.2.2: Prediction plot and error histogram of error function $J = |y - \hat{y}|^2 + 0.5(|y_{capm} - r_f - \hat{y}|)^2$

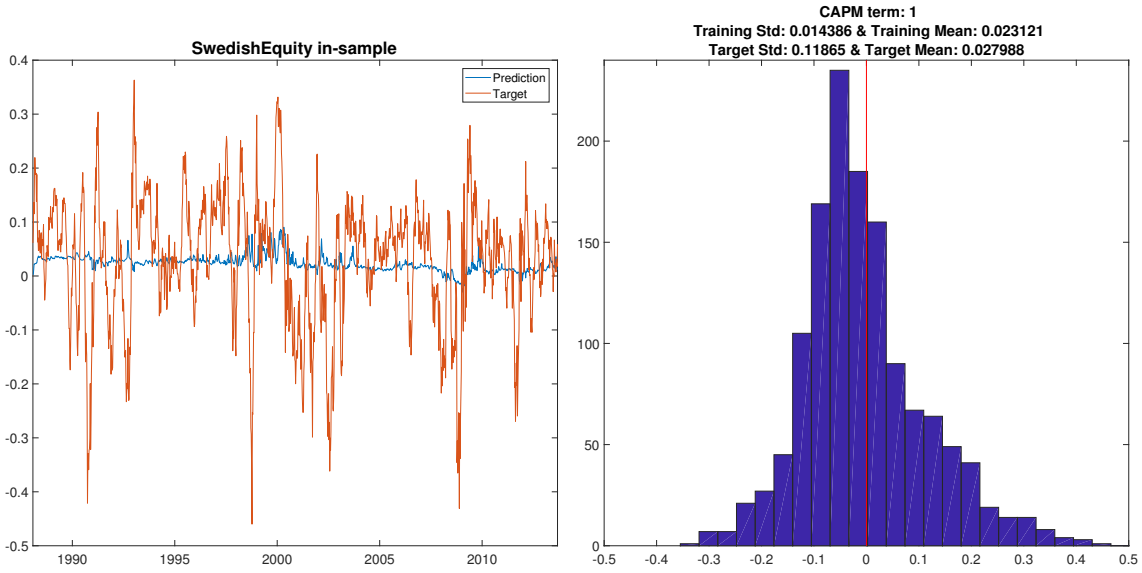

Figure E.2.3: Prediction plot and error histogram of error function $J = |y - \hat{y}|^2 + (|y_{capm} - r_f - \hat{y}|)^2$

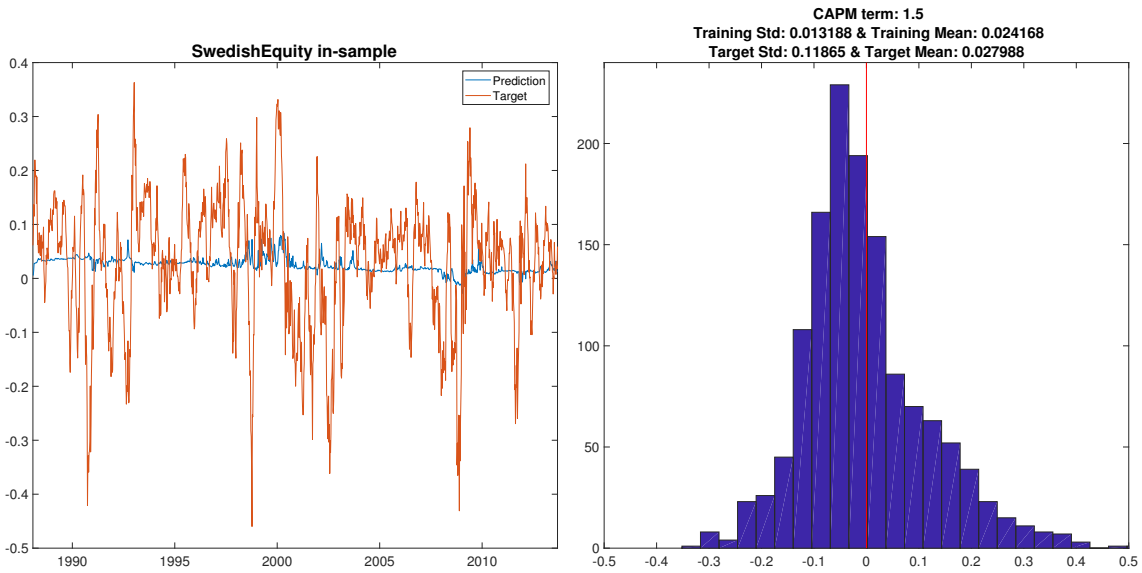

Figure E.2.4: Prediction plot and error histogram of error function $J = |y - \hat{y}|^2 + 1.5(|y_{capm} - r_f - \hat{y}|)^2$

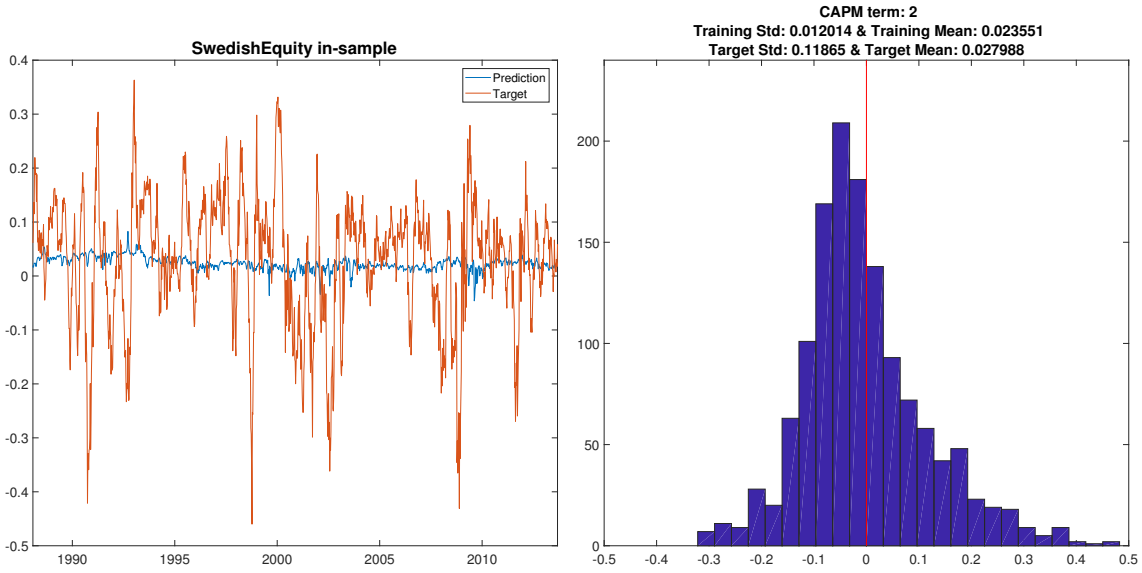

Figure E.2.5: Prediction plot and error histogram of error function $J = |y - \hat{y}|^2 + 2(|y_{capm} - r_f - \hat{y}|)^2$

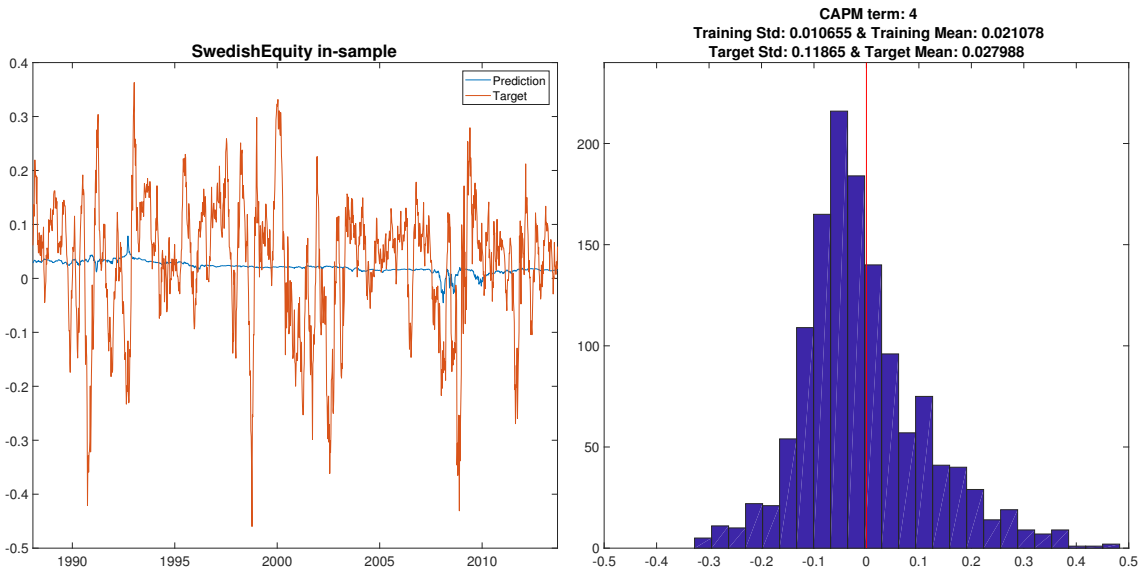

Figure E.2.6: Prediction plot and error histogram of error function $J = |y - \hat{y}|^2 + 4(|y_{capm} - r_f - \hat{y}|)^2$

Appendix F

Resulting networks

Table F.0.1 presents the hyper-parameter-optimized neural networks. The "# pot. inp." field refer to the number of potential inputs, the "# PC red. inp." field refer to the number of inputs where the null-hypothesis of the Peasron correlation test could be rejected, "# Exp. degr." field refer to the chosen explanation degree of the PCA which gave the corresponding number of eigenvectors "# Eig.vec.", and the "Architecture" field refer to the chosen architecture where (x), (x,y), (x,y,z) refer to one, two and three hidden layers respectively, with x,y,z number of hidden neurons in each hidden layer. The $\lambda_{CAPM/RW}$ column contain the CAPM/RW regularization coefficient.

Table F.0.1: Properties of the hyper-paramter-optimized neural networks.

BBG Ticker	**Points**	**# pot. inp.**	**# PC red. inp.**	**Exp. degr.**	**# Eig.vec.**	**Architecture**	$\lambda_{CAPM/RW}$
OMX	1337	32	19	1.000	19	'(5)'	1.0
MXWO	1337	32	25	1.000	25	'(12,1)'	1.1
RXBO	829	52	35	0.975	23	'(4,1)'	2.0
RXRE	826	53	31	0.995	27	'(12,5)'	1.5
RXVX	829	52	48	1.000	48	'(11,3)'	0.0
HE00	763	55	41	0.995	32	'(10,1)'	1.5
LEGATRUU	1176	37	23	1.000	23	'(8,1)'	4.0
JGENBDUU	502	63	31	0.995	23	'(11,4)'	1.7
MXWO0RE	920	46	28	0.995	25	'(10,1)'	1.9
BCOMTR	1128	40	25	0.995	23	'(8,2)'	1.1
HFRXM	755	58	40	1.000	40	'(1)'	2.0
HFRXAR	759	56	38	1.000	38	'(3,4)'	1.6
USDSEK	2171	21	13	0.900	6	'(1,4,1)'	0.6
EURSEK	1233	34	21	1.000	21	'(8,1)'	0.5
JPYSEK	2171	21	13	0.900	7	'(12,5)'	0.1

Table F.0.2: Used inputs to the PCA for index 1-8 including Pearson correlations. The portfolio constituents are presented in the columns, and the potential inputs in the rows. A number in a cell means that the input is used as an input to the network, and the number corresponds to the Pearson correlation. The inputs that were not used as inputs are marked with "-".

Input	OMX	MXWO	RXBO	RXRE	RXVX	HE00	LEGATRUU	JGENBDUU
SwedishEquity	-	-	-0.20	-0.06	0.10	0.19	-0.11	-0.08
GlobalEquity	0.15	-	-0.18	-0.12	0.10	-	-	-0.11
InterestRate	-	-	-	-	-0.06	0.17	-	0.11
Inflation	-	-	-	-	-	-	-	0.12
Cash	-	-	-0.11	-	-	0.10	-	-
Credit	-	-	-	-	-	-	-	-0.11
GlobalBond	-	-	0.10	-	-0.14	-0.06	-	-
EmergingMarketCurrencyBond	-	-	-	-	-	-	-	-
RealEstate	-	-	-0.21	-0.06	0.16	-	-	-0.15
Commodity	-	-	-0.20	0.07	0.17	-	-	-0.14
CTA	-	-	-	-	-	-	-	-
AbsoluteReturn	-	-	-	-	-	-	-	-
JPYSEK	0.16	0.05	0.12	-	-0.17	-	0.16	0.08
EURSEK	-	-	0.07	-0.09	-	-0.08	0.15	-
USDSEK	0.09	-	-	-0.10	-0.18	-0.13	0.21	0.11
1MReturnLag1M	0.10	-	0.14	-0.06	0.32	0.19	0.12	-
1MReturnLag2M	0.05	0.05	0.06	-	0.25	0.12	0.05	-
1MReturnLag3M	-	0.08	-	0.06	0.25	0.15	-	-
1MReturnLag4M	-	0.08	-0.08	0.07	0.26	0.09	-0.05	-
1MReturnLag5M	-	-	-0.10	-	0.30	-	-	-
1MReturnLag6M	-	-	-0.12	-0.09	0.31	-0.11	-	-
1MReturnLag7M	-	0.05	-0.09	-0.07	0.30	-0.08	-	-0.09
1MReturnLag8M	0.07	0.08	-0.06	-	0.30	-	-0.07	-0.16
1MReturnLag9M	0.07	0.09	-	-	0.31	-	-	-0.17
1QReturnLag1Q	0.09	0.09	0.11	-	0.42	0.23	0.10	-
1QReturnLag2Q	-	0.05	-0.16	-	0.40	-	-0.07	-
1QReturnLag3Q	0.09	0.12	-0.08	-	0.38	-	-0.09	-0.25
Inkopschefsindex	0.06	0.06	0.11	0.19	0.24	-0.06	-0.05	-
USProduction	0.08	0.12	-	-	0.19	-0.17	-	-0.14
JPYIndustrialProduction	-	-0.08	-	0.09	-	-0.27	0.05	-
ChinaProduction	-	-	-	0.16	-	-	-	-

Table continues on next page

Input	OMX	MXWO	RXBO	RXRE	RXVX	HE00	LEGATRUU	JGENBDUU
EURProduction	-	-	-	0.06	0.21	-	-	-0.13
Export	-	-	-	-	-	-	-	-
USConsumerConfidence	-	-	-0.06	-	0.48	-0.26	-	-
GDPUS	-	0.05	0.06	0.13	0.40	-0.25	0.07	-
EURGDP	-	-	-0.12	-	0.28	-0.36	-	-
SWEGDP	-0.18	-0.25	-	-	0.11	-0.35	-0.12	-0.13
CHGDP	-	-	-	0.07	-0.37	-0.23	-	-
JPYGDP	-	-	-0.14	-	0.08	-0.07	-	-
UnemploymentUS	0.06	0.07	0.10	0.08	-0.32	0.31	-	-
EURUnemployment	-	-	-	-	-	0.31	-	-0.15
CHUnemployment	-	-	-	-	-	-	-	-
JPYUnemployment	-	-	-	-	-	-	-	-
USMoneySupply	-	0.06	0.08	-	-0.37	0.31	-	-
CHMoneySupply	-	-	0.07	0.24	0.18	-	-	-
EURMoneysupply	0.24	0.19	-0.12	-	0.52	0.32	-0.15	-
JPYMoneySupply	-	-	-	-	-	-	-	-0.16
USInflation	-0.15	-0.12	0.11	-	-0.11	-0.51	0.27	0.18
JPYInflation	-0.07	-0.05	0.12	-0.09	-0.24	-0.25	0.16	0.10
EURInflation	-	-	0.13	-	-0.26	-0.44	-	0.10
CHInflation	-	-	0.09	0.10	-0.43	-0.26	-	-
BalticDryIndex	-0.14	-0.14	-	0.07	-0.16	-0.21	-	0.10
SWEKonjunkturbarometern	-	-	-	0.06	0.11	-0.26	-	-0.13
PEMSCIWorld	-	-	-	0.11	0.49	-	-	0.10
US10YYields	-	-0.06	-	0.12	0.44	-0.29	0.11	0.14
US3YYields	-	-0.05	-	-	0.36	-0.30	0.10	0.10
US3MYields	-	-	-0.10	-0.07	0.31	-0.26	-	-
JPY10YYields	-0.06	-0.07	-	0.17	0.13	-0.20	0.08	0.12
JPY3YYields	-	-	-	-	-0.11	-0.23	0.08	-
JPY3MYields	-0.10	-0.10	0.09	-0.07	-0.39	-0.15	0.09	0.08
EUR10YYields	-	-	0.17	0.22	0.39	-0.31	-	0.18
EUR3YYields	-	-	0.11	0.13	0.33	-0.41	-	0.14
EUR3MYields	-	-	-	-	-	-	-	0.13
CN10YYields	-	-	-	-	-	-	-	-
CN3YYields	-	-	-	-	-	-	-	-
CN3MYields	-	-	-	-0.08	-	-0.20	-	-

Table F.0.3: Used inputs to the PCA for index 9-15 including Pearson correlations. The portfolio constituents are presented in the columns, and the potential inputs in the rows. A number in a cell means that the input is used as an input to the network, and the number corresponds to the Pearson correlation. The inputs that were not used as inputs are marked with "-".

Input	MXWO0RE	BCOMTR	HFRXM	HFRXAR	JPYSEK	EURSEK	USDSEK
SwedishEquity	0.06	0.14	-0.14	-	-	-0.22	-
GlobalEquity	-	0.11	-	-	-	-0.13	-
InterestRate	-	-	0.14	0.11	-	-	-
Inflation	-	-	0.07	0.10	-	-	-
Cash	-	-	0.07	0.13	-	-	-
Credit	-	-	-0.22	-0.07	-	-	-
GlobalBond	-0.10	-0.08	0.15	-	-	-	-
EmergingMarketCurrencyBond	-	-	-	-	-	-	-
RealEstate	-	-	-0.09	-	-	-	-
Commodity	0.11	-	-0.09	-	-	-	-
CTA	-	-	-	-	-	-	-
AbsoluteReturn	-	-	0.12	-	-	-	-
JPYSEK	-0.13	-	0.14	-	-	0.11	0.04
EURSEK	-0.09	-0.11	0.12	-	-	-	-
USDSEK	-0.09	-0.09	0.14	0.06	0.16	0.12	-
1MReturnLag1M	-	-	-	-	0.15	0.05	0.07
1MReturnLag2M	-	0.08	-	-	0.10	-	0.05
1MReturnLag3M	0.07	0.06	-	0.09	0.05	-	0.06
1MReturnLag4M	0.07	-	-	0.08	-	-0.08	-
1MReturnLag5M	-	-	-	0.07	-0.06	-0.05	-
1MReturnLag6M	-	-	-0.07	-	-0.05	-	-0.04
1MReturnLag7M	0.13	0.07	-	-0.07	-	-	-
1MReturnLag8M	0.14	-	-	-0.14	-	-	-
1MReturnLag9M	-	-	-	-0.08	-	-	-
1QReturnLag1Q	-	0.08	0.08	0.08	0.16	0.06	0.10
1QReturnLag2Q	-	-	-	0.09	-0.06	-0.07	-
1QReturnLag3Q	0.18	-	-	-0.17	-	-	-
Inkopschefsindex	-	0.08	-	-	-	-	-0.11
USProduction	-	0.10	0.14	0.15	-	-	-0.05
JPYIndustrialProduction	-0.08	0.09	0.17	0.18	-	-0.05	-
ChinaProduction	0.06	0.06	-0.12	-0.09	-	-	-

Table continues on next page

Input	MXWO0RE	BCOMTR	HFRXM	HFRXAR	JPYSEK	EURSEK	USDSEK
EURProduction	0.06	-	-	-	-	-	-
Export	-	-	-	-	-	-	-
USConsumerConfidence	-	0.13	0.27	0.24	-	-	-
GDPUS	-	0.17	0.27	0.27	0.06	0.11	0.04
EURGDP	-0.06	-	0.20	0.20	-	-	-
SWEGDP	-0.20	-	0.16	0.11	-	-	-
CHGDP	-0.08	-	-	-0.07	-	-	-
JPYGDP	0.15	0.21	-	0.10	-	-0.05	-
UnemploymentUS	0.10	-0.06	-0.29	-0.24	0.12	-0.13	0.08
EURUnemployment	-	-	-0.12	-	-	-	-
CHUnemployment	-	-	-	-	-	-	-
JPYUnemployment	-	-	-	-	-	-	-
USMoneySupply	0.07	-0.07	-0.24	-0.18	-	-	-
CHMoneySupply	-	0.14	-0.06	-	-	-	-
EURMoneysupply	0.22	0.27	-	-	-	-0.15	-
JPYMoneySupply	-	-	-	-	-	-	-
USInflation	-0.11	-0.07	0.39	0.34	0.07	0.18	0.15
JPYInflation	-0.28	-0.18	0.20	0.14	-	0.17	-
EURInflation	-	-	0.17	0.09	-	-	-
CHInflation	-	-	-	-	-	-	-
BalticDryIndex	-0.16	-	-	-	-	-	-
SWEKonjunkturbarometern	-	-	0.10	0.11	-	-	-
PEMSCIWorld	-	-	0.11	0.09	-	-	-
US10YYields	-	0.10	0.22	0.19	0.14	0.12	0.14
US3YYields	-	0.10	0.27	0.26	0.11	0.11	0.14
US3MYields	-	-	0.28	0.27	-	-	-
JPY10YYields	-	0.07	0.14	0.12	-	0.11	-
JPY3YYields	-0.10	-	0.11	0.11	-	0.11	-
JPY3MYields	-0.26	-	0.15	-	-	0.11	-
EUR10YYields	-0.11	-	0.19	0.13	-	-	-
EUR3YYields	-0.18	-	0.25	0.19	-	-	-
EUR3MYields	-	-	0.33	-	-	-	-
CN10YYields	-	-	-	-	-	-	-
CN3YYields	-	-	-	-	-	-	-
CN3MYields	-	-	0.24	0.23	-	-	-

Appendix G

Out-of-sample performance

G.1 Plots of out-of-sample predictions

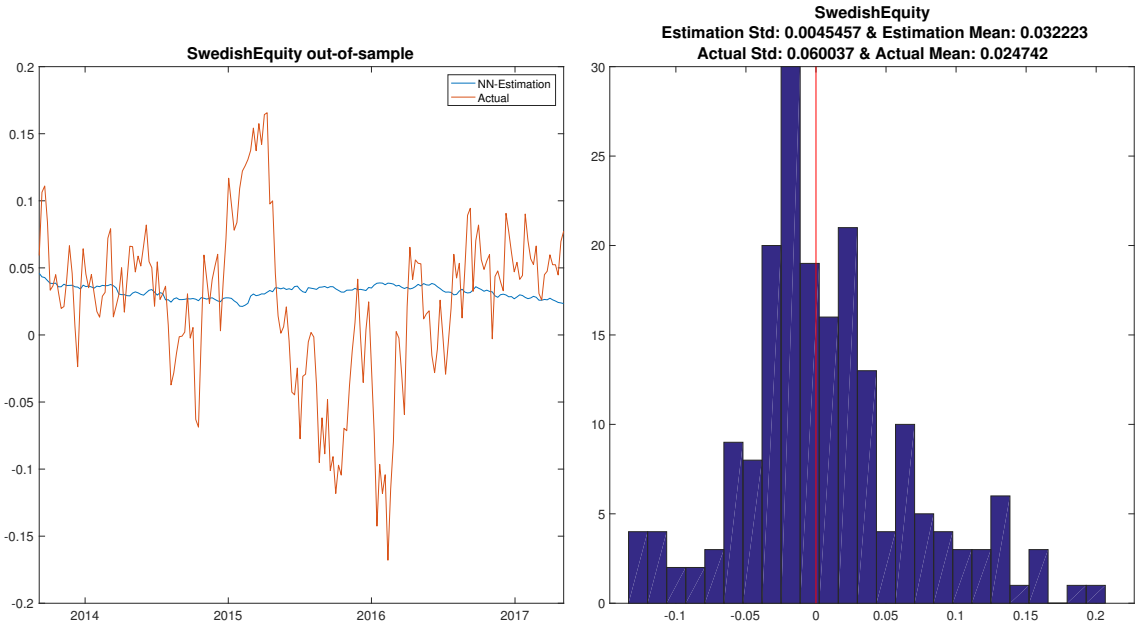

Figure G.1.1: OMX Index

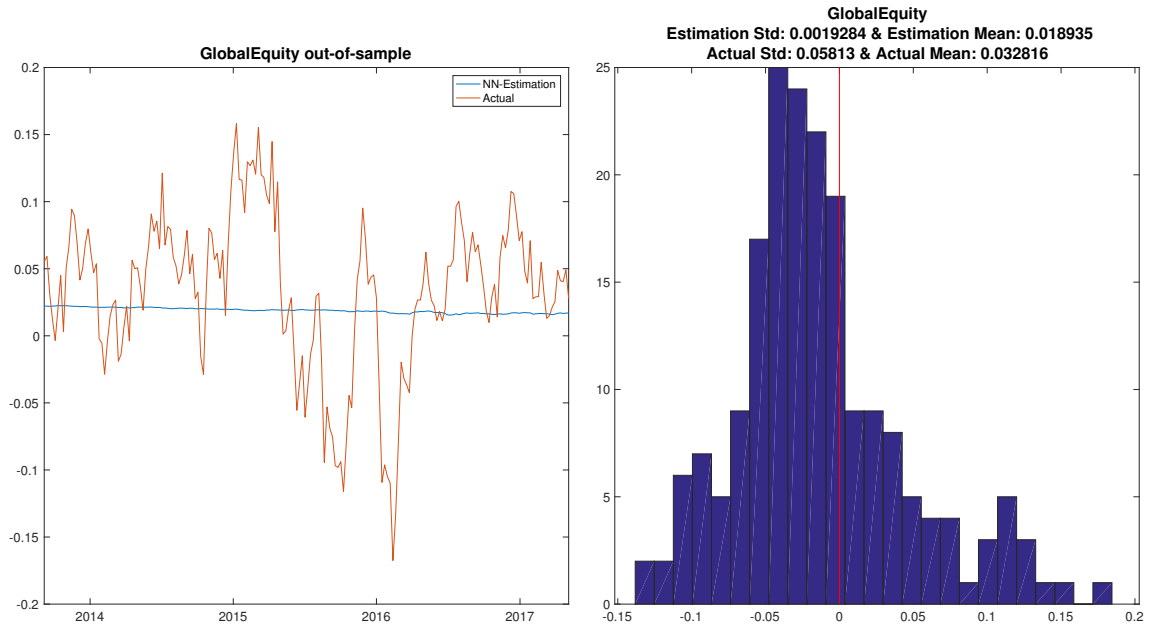

Figure G.1.2: MXWO Index

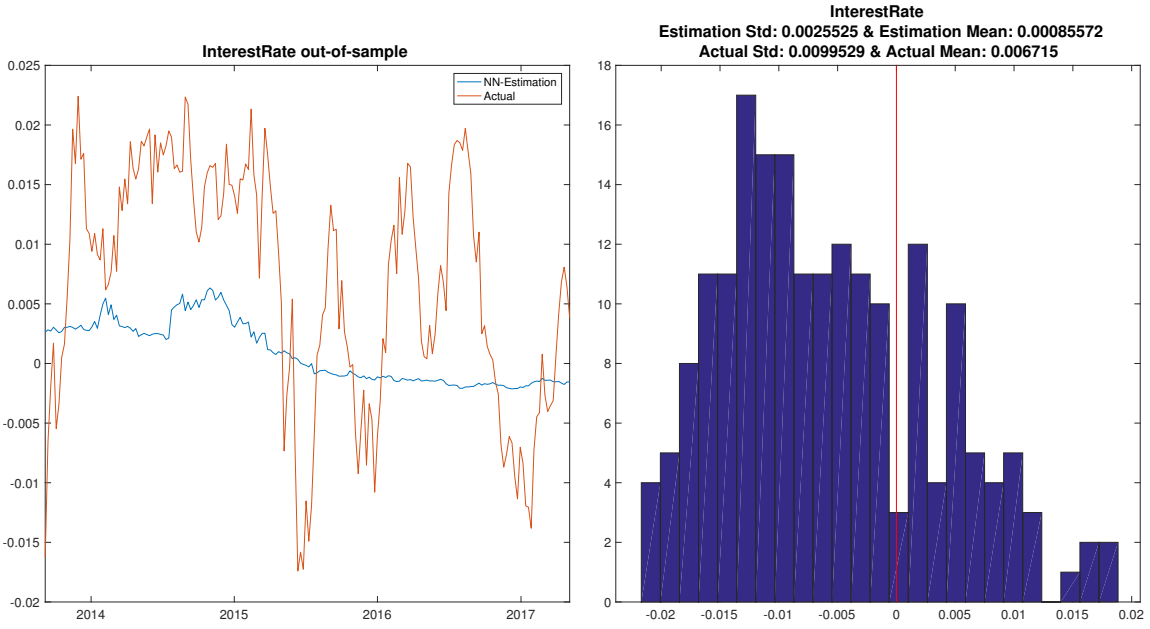

Figure G.1.3: RXBO Index

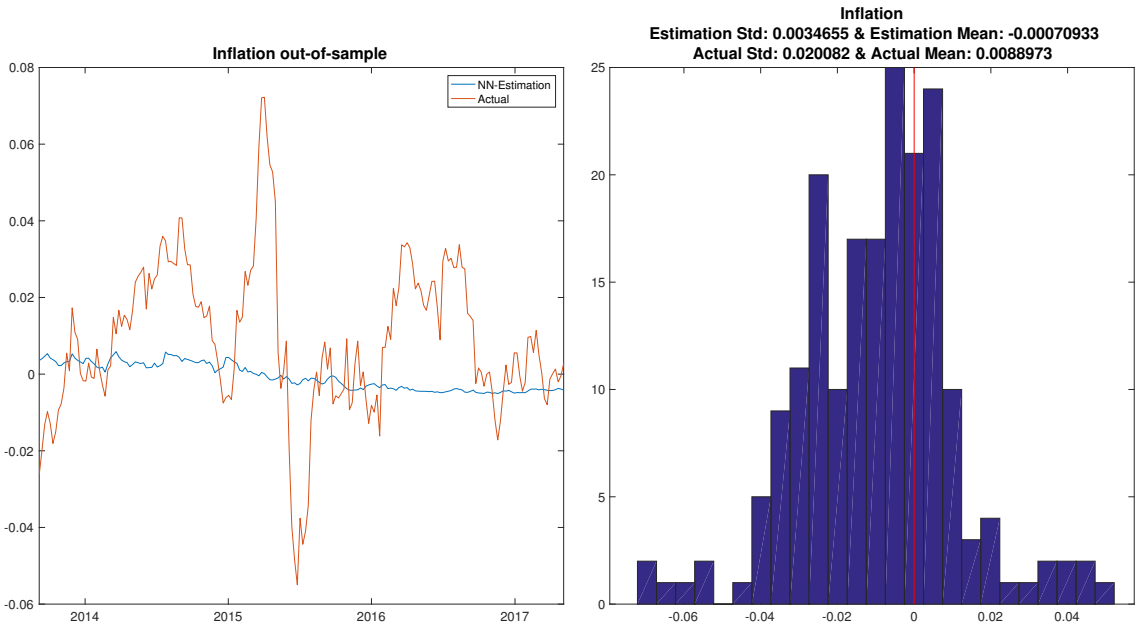

Figure G.1.4: RXRE Index

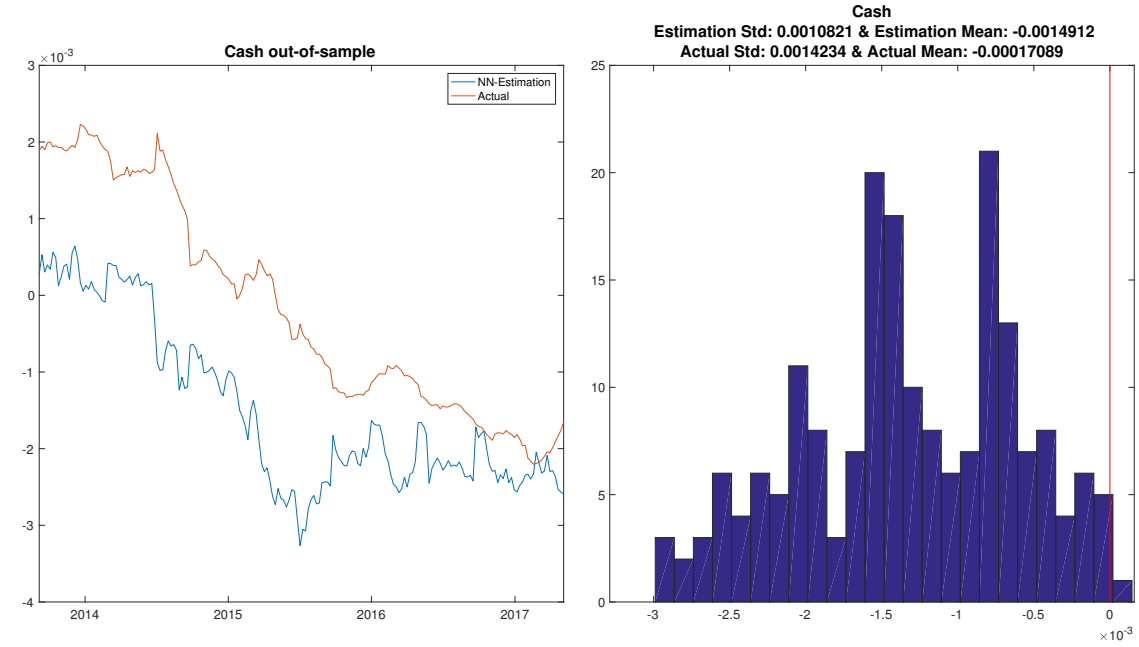

Figure G.1.5: RXVX Index

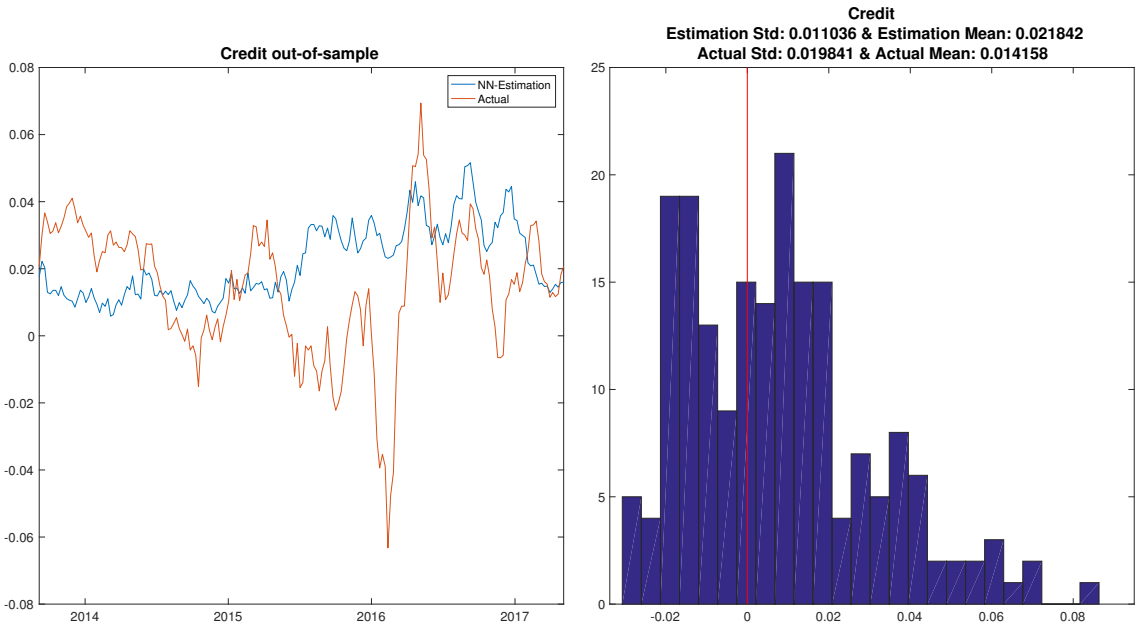

Figure G.1.6: HE00 Index

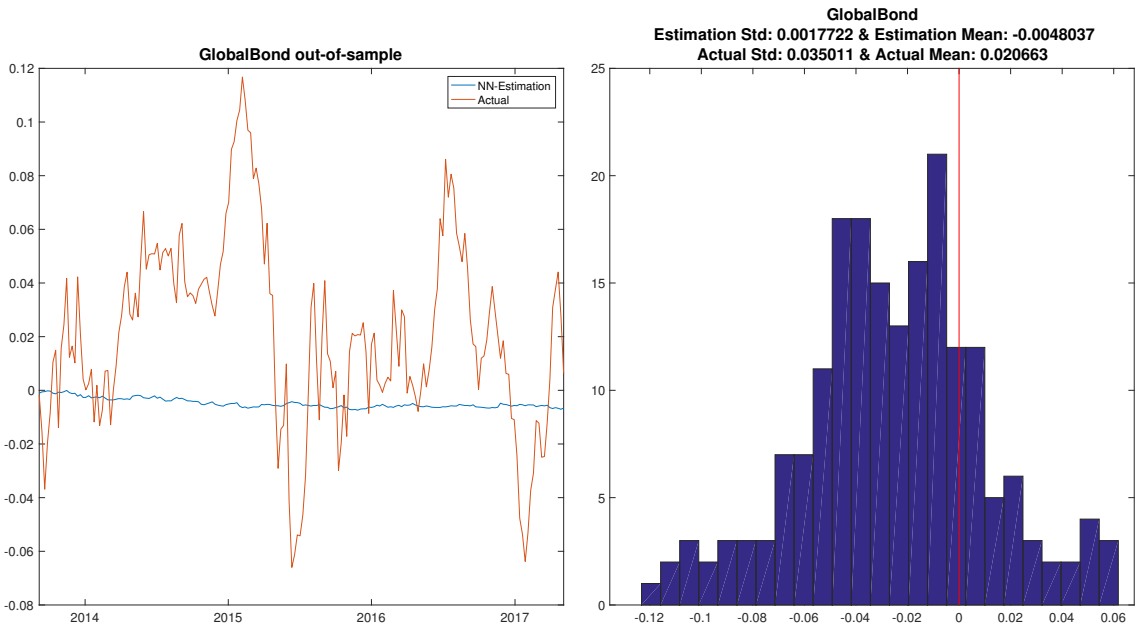

Figure G.1.7: LEGATRUU Index

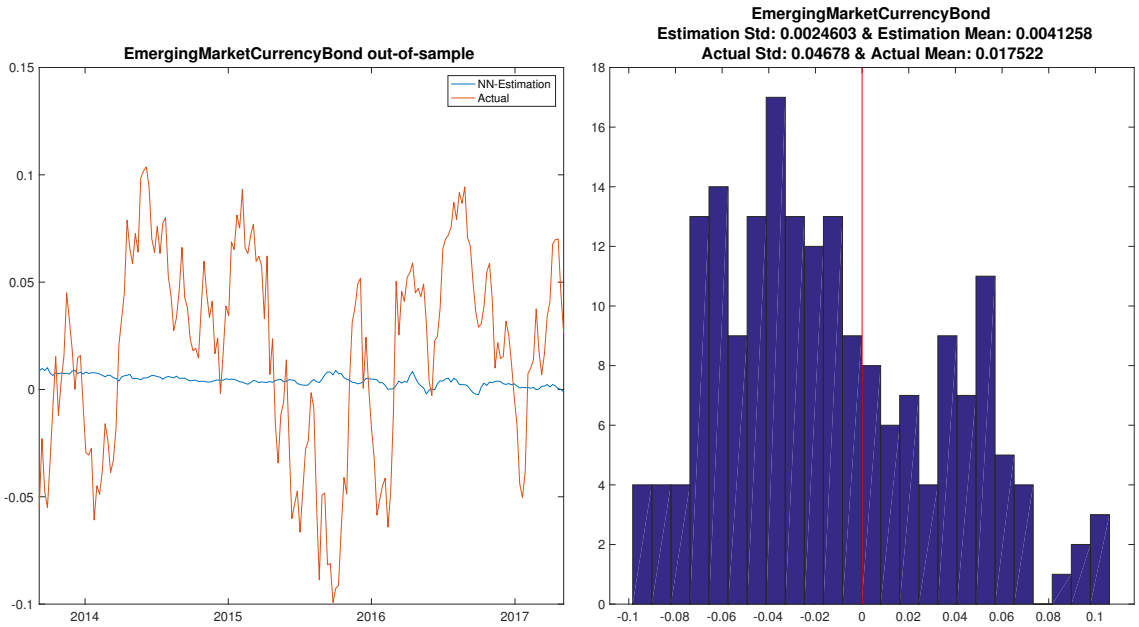

Figure G.1.8: JGENBDUU Index

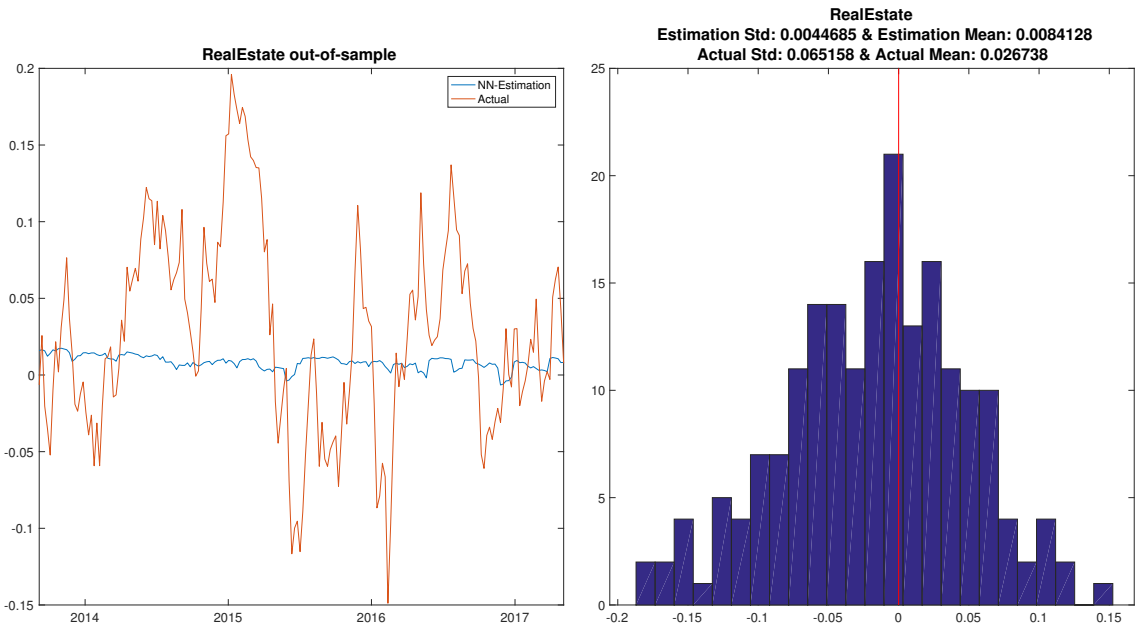

Figure G.1.9: MXWO0RE Index

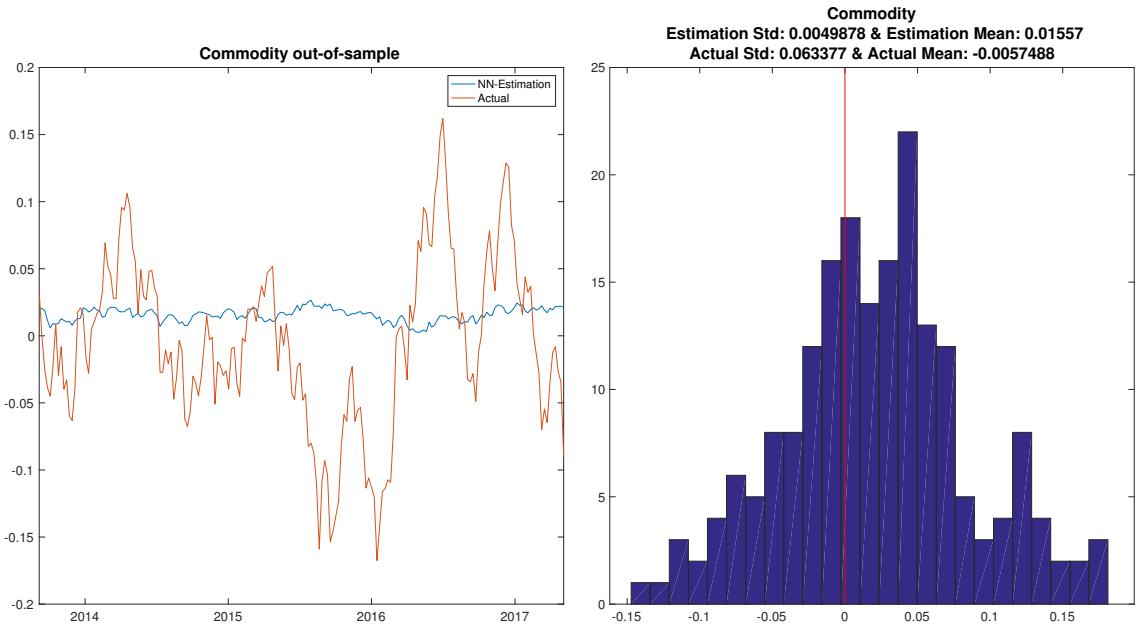

Figure G.1.10: BCOMTR Index

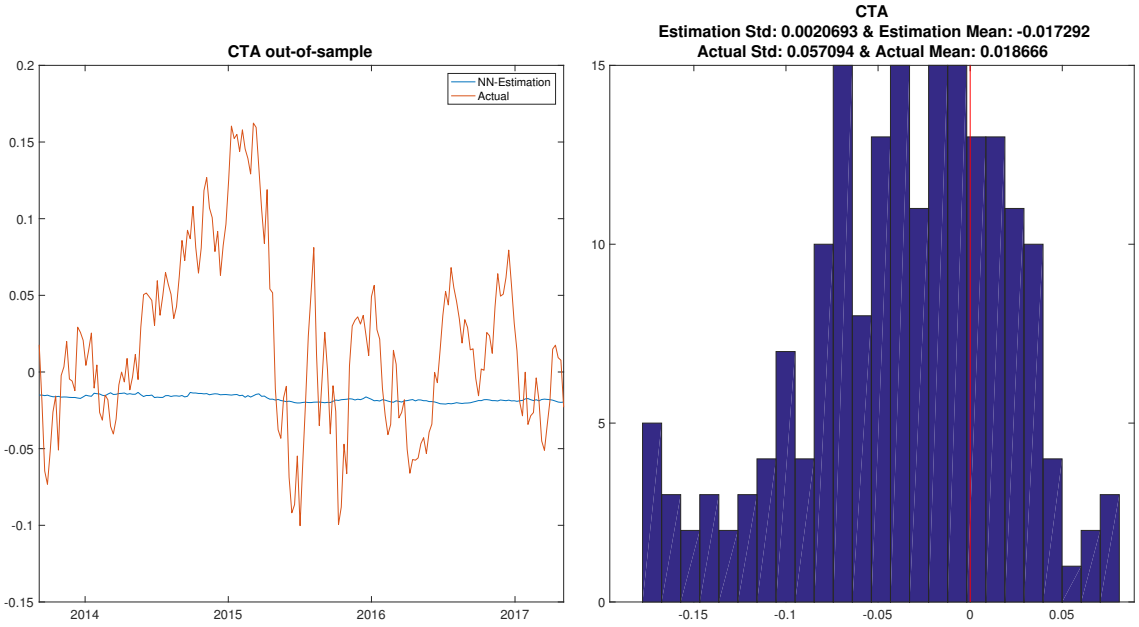

Figure G.1.11: HFRXM Index

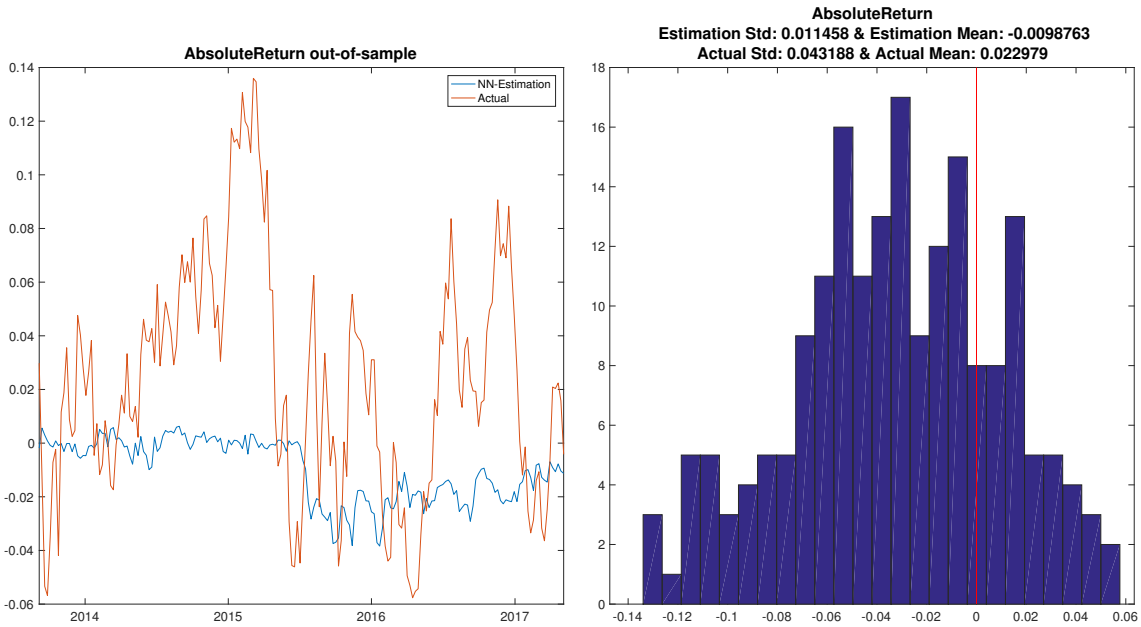

Figure G.1.12: HFRXAR Index

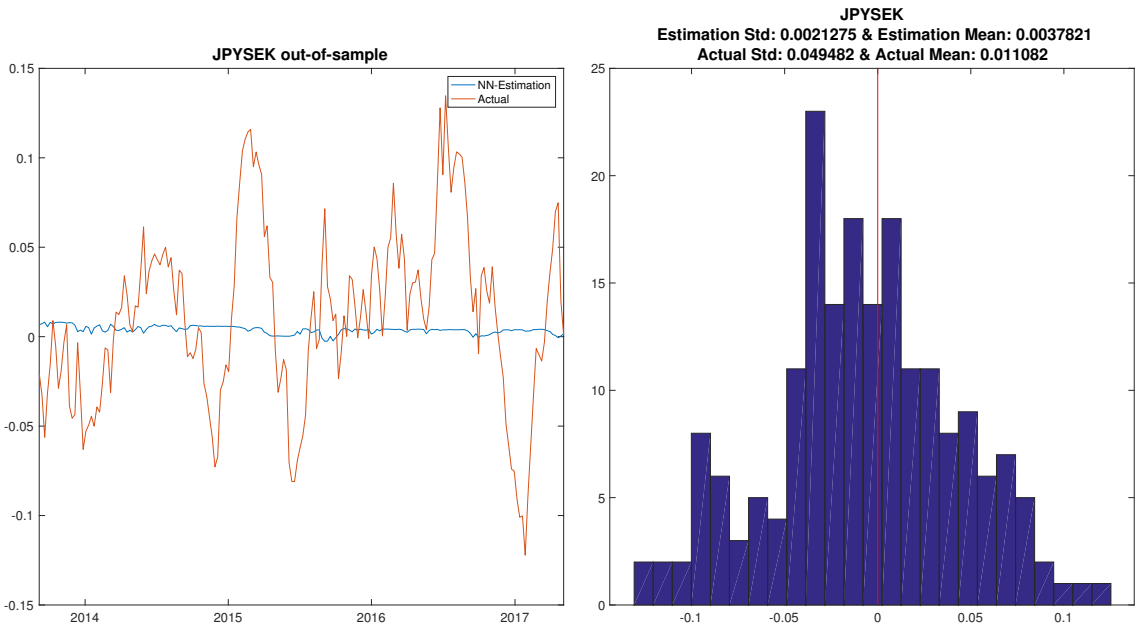

Figure G.1.13: JPYSEK Index

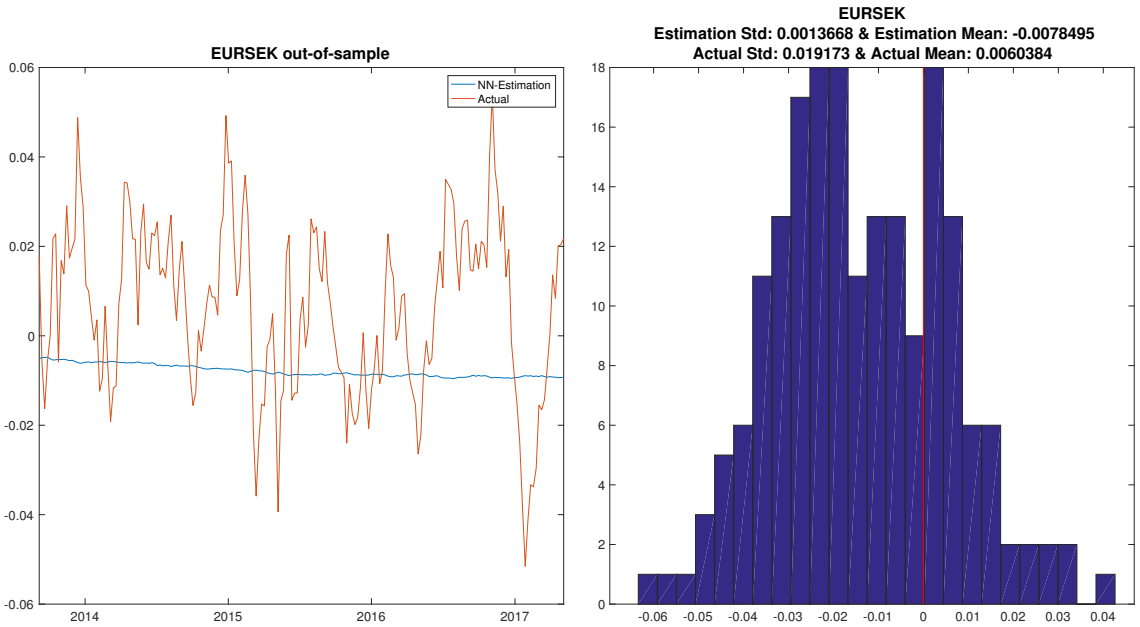

Figure G.1.14: EURSEK Index

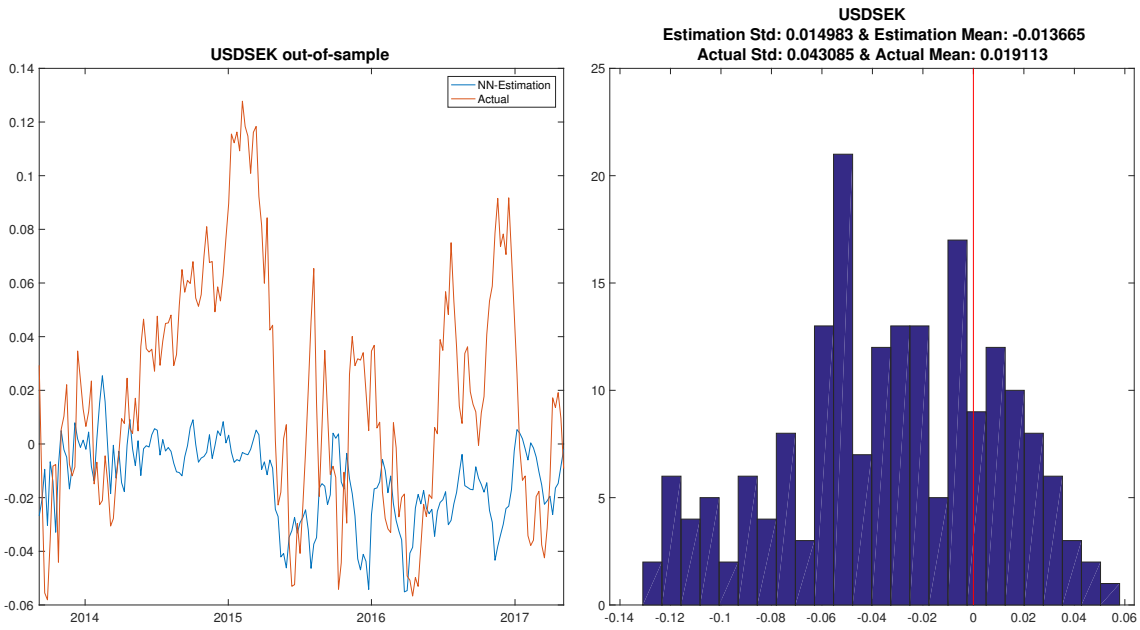

Figure G.1.15: USDSEK Index