universität wien

MASTERARBEIT / MASTER'S THESIS

Titel der Masterarbeit / Title of the Master's Thesis

„Volatility forecasting with Neural Networks“

verfasst von / submitted by

Anela Jahic

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Master of Science (MSc)

Wien, 2020 / Vienna 2020

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	UA 066 821
Studienrichtung lt. Studienblatt / degree programme as it appears on the student record sheet:	Masterstudium Mathematik
Betreut von / Supervisor:	Dipl.-Ing. Dr. Christa Cuchiero, Privatdoz.

Abstract

In this thesis volatility forecasting of financial assets is studied. We consider first the time series of returns and present its main characteristics and stylized facts. The behaviour of the variance of return series is explained, and the time series models for conditional volatility forecasting are presented. Then, we consider a multilayer artificial neural network and present a back propagation algorithm. In the empirical study, we test artificial neural networks for realized volatility forecasting of the monthly returns of S&P 500 and NASDAQ stock indexes. As a benchmark for the results obtained by artificial neural networks, ARCH models are tested on the same problems. ARCH models are ARCH (1,1), GARCH (1,1) and EGARCH(1,1).

Key words: Realized Volatility Forecasting; Return Time Series; Artificial Neural Networks; GARCH Models

Abstrakt

In dieser These werden sprunghafte / unbeständige Vorhersagen von finanziellen Vermögenswerten untersucht. Wir betrachten zunächst die Zeitreihe der Returns und stellen ihre Hauptmerkmale und stilisierten Fakten vor. Das Verhalten der Varianz von Return-Reihen wird erklärt und die Zeitreihenmodelle für die bedingte Volatilitätsvorhersage werden vorgestellt. Dann betrachten wir ein mehrschichtiges künstliches neuronales Netz und stellen einen Rückpropagationsalgorithmus vor. In der empirischen Studie testen wir künstliche neuronale Netze zur realisierten Volatilitätsprognose des monatlichen Returns von S&P 500 und NASDAQ-Aktienindizes. Als Maßstab für die mit künstlichen neuronalen Netzen erzielten Ergebnisse werden ARCH-Modelle an denselben Problemen getestet. ARCH-Modelle sind ARCH (1,1), GARCH (1,1) and EGARCH(1,1).

Schlüsselwörter: Realisierte Volatilitätsvorhersage; Return-Zeitreihen, Künstliche Neuronale Netze, GARCH-Modelle

Acknowledgments

I would like to express my gratitude and appreciation to my supervisor Dipl.-Ing. Dr. Christa Cuchiero, Privatdoz. for her assistance and constructive suggestions during the development of this thesis.

I want to thank to my parents, Suzana and Dzevad Jahic, and to my boyfriend Emir, for their love, support and encouragement.

Contents

Introduction

Prediction of future volatility movements of financial assets has been analyzed for decades. Future volatility of stocks, cash, bonds, etc. is usually estimated based on historical data such as times series of prices. This problem is tackled within the field of financial time series analysis and a large number of models has been developed so far.

In the early eighties Engel [8] developed the so-called auto regressive conditional heteroscedastic (ARCH) model which was the basis for further development and generalizations. A few years later Borellslev [5] introduced generalized ARCH (GARCH) models which are nowadays widely in use. These models are known to be reliable and are often used in financial institutions for volatility modeling of financial assets.

Artificial neural network algorithms are dating back to 1960s. Since then various types of networks have been developed and algorithms are improved using novel optimization methods. Artificial neural networks become increasingly popular in various fields of the science, but are still not standard in financial forecasting.

The goal of the thesis is to investigate how well artificial neural networks perform when implemented for non-linear prediction problems. The main focus of the thesis is the structure and the performance of neural networks which are used for volatility forecasting of financial assets. The empirical study is centered around the future prediction of the realized volatility for the closing prices of S&P 500 and NASDAQ stock indexes. In order to evaluate the result obtained by neural networks, we implemented GARCH models on the same problems and compared the results.

The thesis is divided into three main chapters. Chapter 1 explains and describes financial time series. In this chapter, a definition of volatility is presented and GARCH models are explained. Artificial neural networks are introduced in Chapter 2 together with their structure and the back propagation algorithm. The implementation of both models is presented in Chapter 3. This chapter contains the data description, results of the models and discussion on the results.

The empirical part of this thesis is done in the programming language Python, version 3.7(64-bit). Neural networks are implemented in Keras (high level neural networks application programming interface), while GARCH models are done with the ARCH toolbox (tool for financial econometrics in Python).

Chapter 1

Financial time series

1.1 Return

Return is defined as a profit or loss of a portfolio. A portfolio is a collection of financial assets, such as stocks, bonds, cash and cash equivalents, or other investments. Return at the time t, $t \geq 0$ of a single asset can be expressed as a simple gross return

$$1 + R_t = \frac{P_t}{P_{t-1}}, \tag{1.1}$$

where P_t is the price of the underlying asset. From equation (1.1), a simple net return can be obtained and is given by formula

$$R_t = \frac{P_t - P_{t-1}}{P_{t-1}}. \tag{1.2}$$

The return of the whole portfolio is a weighted sum of single returns at the time t given by

$$R_{p,t} = \sum_{i=1}^{N} w_i R_{it} \tag{1.3}$$

where N is the number of assets in the portfolio, R_{it} is the net return of individual assets, w_i is the fraction of the portfolio's value in that asset.

The return r_t is usually given as a *continuously compounded return* or a *logarithmic return*. It is calculated as a natural logarithm of the simple gross return such that

$$r_t = \ln(1 + R_t) = \ln(\frac{P_t}{P_{t-1}}). \tag{1.4}$$

1.2 Return Time Series

A *time series* is a stretch of values on the same scale indexed by a time-like parameter [6].

In order to define a financial return time series, we will first introduce a couple of definitions. The definitions can be found in [20].

Definition 1 *Let Ω be a nonempty set, let $\mathcal{F}$ be a collection of subsets of Ω. $\mathcal{F}$ is a* σ-algebra *if the following is satisfied:*

(i) $\emptyset \in \mathcal{F}$

(ii) for each set A, if $A \in \mathcal{F}$, then $A^c \in \mathcal{F}$

(iii) for each sequence of the sets $A_1, A_2, ...$ such that $A_1, A_2, ... \in \mathcal{F}$, then $\bigcup_{n=1}^{\infty} A_n \in \mathcal{F}$.

Definition 2 *Let Ω be a nonempty set and $\mathcal{F}$ be a $\sigma-$algebra of subsets of Ω. A* probability measure *P is a function that, to every set $A \in \mathcal{F}$, assigns a number in $[0, 1]$, called the* probability *of A and written $P(A)$. It is required that:*

(i) $P(\Omega) = 1$

(ii) if $A_1, A_2, ...$ is a sequence of disjoint sets in $\mathcal{F}$, then

$$P(\bigcup_{n=1}^{\infty}) = \sum_{n=1}^{\infty} P(A_n), \tag{1.5}$$

Then, the triple $(\Omega, \mathcal{F}, P)$ is called a probability space.

Definition 3 *Let $(\Omega, \mathcal{F}, P)$ be a probability space and $I \in \mathbb{R}$ is an index set. Assume that for each $\alpha \in I$, there is a random variable $X_\alpha : \Omega \to \mathbb{R}$ defined on $(\Omega, \mathcal{F}, P)$. The function $X : I \times \Omega \to \mathbb{R}$ defined by $X(\alpha, \omega) = X_\alpha(\omega)$, $\omega \in \Omega$, is called a real-valued* stochastic process *with index set I and it is written $\{X_\alpha\}_{\alpha \in I}$.*

Definition 4 *A real-valued stochastic process $\{r_t\}_{t \in [0,\infty)}$ is called* return stochastic process *(or* return time series*) if r_t is the return random variable observed at time t. It is simply a collection of return random variables indexed by time and will be denoted by $\{r_t\}_{t \geq 0}$.*

1.2.1 Stylized Facts about Return Series

Return series obtained from different assets and different markets can exhibit similar statistical properties. Such properties common across the wide range of instruments, markets and time periods are called *stylized empirical facts* [7]. The main facts are explained below.

- The distribution of a probability density function of a time series deviates from a normal distribution, often returns have thicker tails. One way to quantify the deviation from the normal distribution is to calculate kurtosis of the time series distribution. Gaussian distribution has a value of the kurtosis around 0, where a large positive value of the kurtosis defines fat tails. But, as the time scale increases, the return distribution gets closer to the normal distribution [18].

- Linear autocorrelation function of the underlying price changes in asset returns is decaying rapidly and in a short time, the function is getting to zero. The absence of the correlation is explained as if the price changes exhibited significant correlation, this correlation could be used to obtain a strategy with positive earnings, which is known as an arbitrage [7]. This type of strategy tends to reduce the correlation except for a short time period, needed for the market to react to the new information [7]. If the time scale is increasing, weekly and monthly returns are indicating a slightly significant autocorrelation.

- Volatility clustering. This phenomenon was introduced by Mandelbrot [14], who stated: "Large changes tend to be followed by large changes, and small changes tend to be followed by small changes". Volatility clustering is related to the autocorrelation of the return series, that is, while returns r_t, $t \geq 0$, themselves are uncorrelated, their squares or absolute returns are showing a positive, significant and slowly decaying autocorrelation function: $corr(|r_t|, |r_{t+\tau}|) > 0$. Volatility clustering is the tendency of the large changes in prices of financial assets to cluster together. In the practice, this means that when receiving a new information the market is responding with large price movements and these changes last for some time. Volatility clustering allows us to make some predictions about the future movements in the price that are based on the recent changes, although the process is random.

- Heteroscedasticity. Most of the financial (return) time series are having a time-dependent variance. This means that the variance of the return series is not constant and changes over the time as the series evolves.
- Leverage effect. It is empirically observed that volatility and asset returns are negatively correlated. Volatility bursts are more likely related to the negative past returns [2].

1.2.2 Variance of Return Series

Let us consider a sequence of return random variables $\{r_t\}_{t\geq 0}$ where $r_t \in \mathbb{R}$.

Definition 5 *A real-valued stochastic process $\{r_t\}_{t\geq 0}$ is defined to be* covariance *or* weak stationary *if the first moment (expectation) and covariance of the process do not vary with respect to time and the second moment is finite for all times [9].*

The definition implies the following

$$E[r_t] = \mu$$

$$Cov[r_t, r_{t+\tau}] = \gamma(\tau)$$

$$E[|r_t|^2] < \infty.$$

Any covariance stationary process can be expressed as the sum of two processes: deterministic process and infinitive moving average process. That is stated in Wold's decomposition theorem given below [10]:

Theorem 1 *Let the process $\{r_t\}_{t\geq 0}$ be covariance stationary, then the process at time t has the following representation*

$$r_t = \mu_t + \sum_{j=0}^{\infty} b_j \epsilon_{t-j} \tag{1.6}$$

with $b_0 = 1$.

In equation (1.6) μ_t is deterministic part and $\sum_{j=0}^{\infty} b_j \epsilon_{t-j}$ is stochastic part called infinite moving average process and $\sum_{j=0}^{\infty} b_j^2 < \infty$. ϵ_t is a sequence of uncorrelated random variables with mean zero and variance σ^2. ϵ_t is called *white noise*. Processes μ_t and $\sum_{j=0}^{\infty} b_j \epsilon_{t-j}$ are uncorrelated.

Wold's decomposition shows that a return time series, that happens to be a weak stationary process, can be approximated by a linear model. In practice, time series models are more complex with a time-varying variance (non-stationary processes) requiring non-linear models relating the process $\{r_t\}_{t\geq 0}$ to its past observations.

Let us now assume a stronger condition on ϵ_t, namely that the process ϵ_t is i.i.d. (independent identically distributed) with mean zero and variance σ^2. For the sake of simplicity let the μ_t be zero. This reduces the process to the following form

$$r_t = \sum_{j=0}^{\infty} b_j \epsilon_{t-j}. \tag{1.7}$$

The unconditional mean of the process $\{r_t\}_{t\geq 0}$ is the following

$$E[r_t] = 0, \tag{1.8}$$

and the unconditional variance is given by

$$\begin{aligned} Var[r_t] = Var[\sum_{j=0}^{\infty} b_j \epsilon_{t-j}] &= E[(\sum_{j=0}^{\infty} b_j \epsilon_{t-j} - E(\sum_{j=0}^{\infty} b_j \epsilon_{t-j}))^2] \\ &= E[\sum_{j=0}^{\infty} b_j^2 \epsilon_{t-j}^2 + 2\sum_{j<l} b_j b_l \epsilon_{t-j}\epsilon_{t-l}] \\ &= E[\sum_{j=0}^{\infty} b_j^2 \epsilon_{t-j}^2] + 2E[\sum_{j<l} b_j b_l \epsilon_{t-j}\epsilon_{t-l}] \\ &= E[\sum_{j=0}^{\infty} b_j^2 \epsilon_{t-j}^2] = \sum_{j=0}^{\infty} b_j^2 E[\sum_{j=0}^{\infty} \epsilon_{t-j}^2] = \sigma^2 \sum_{j=0}^{\infty} b_j^2. \end{aligned} \tag{1.9}$$

Due to the conditions on b_j, the unconditional variance given in (1.9) is finite.

Now, let us investigate how the conditional mean and the variance of the process defined in (1.7) behave. Before doing so, we shall introduce the following definitions.

Definition 6 *Let Ω be a nonempty set. Let T be a fixed positive number and $T \in \mathbb{N}$, and assume that for each $t \in [0,T]$ there is a $\sigma-$algebra $\mathcal{F}(t)$. Assume further that if $s \leq t$, then every set in $\mathcal{F}(s)$ is also in $\mathcal{F}(t)$. Then the collection of $\sigma-$algebras $\mathcal{F}(t)$ where $t \in [0,T]$ is a* filtration *[20].*

The σ-algebra $\mathcal{F}(t)$ represents the information available at time t.

A real-valued random variable r_t is said to be $\mathcal{F}(t)$-*measurable* if the information in $\mathcal{F}(t)$ at time t is sufficient to determine the value of r_t. [20].

Definition 7 *Let $\{\epsilon_t\}_{0\leq t\leq T}$ be a collection of random variables and $\mathcal{F}(t)$ is a filtration. This collection of random variables is an* adapted stochastic process*, if for each t, the random variable ϵ_t is $\mathcal{F}(t)$-measurable. [20].*

The conditional mean and the variance depending on the information up to time t of the process given by (1.7) are the following

$$E[r_t|\mathcal{F}_{t-1}] = \sum_{j=1}^{\infty} b_j \epsilon_{t-j} \tag{1.10}$$

where $E[b_0\epsilon_t|\mathcal{F}_{t-1}] = 0$ since $b_0 = 1$,

and

$$\begin{aligned} Var[r_t|\mathcal{F}_{t-1}] &= E[(r_t - E(r_t|\mathcal{F}_{t-1}))^2|\mathcal{F}_{t-1}] \\ &= E[(\sum_{j=0}^{\infty} b_j\epsilon_{t-j} - \sum_{j=1}^{\infty} b_j\epsilon_{t-j})^2|\mathcal{F}_{t-1}] \\ &= E[\epsilon_t^2|\mathcal{F}_{t-1}] = \sigma^2, \end{aligned} \tag{1.11}$$

with the process $\{\epsilon_t\}_{t\geq 0}$ adapted to filtration $\mathcal{F}_{t-1}$.

From (1.11) it can be seen that the conditional variance, depending on the previous observations, of the process that is assumed to be weak stationary is constant. This is opposite to the presence of the heteroscedasticity that is empirically observed in the financial return series.

Even more, let us consider the conditional prediction error variance at the step $k+1$ which is given as follows

$$E[r_{t+k}|\mathcal{F}_t] = \sum_{j=1}^{\infty} b_{j+k}\epsilon_{t-j}, \tag{1.12}$$

$$r_{t+k} - E[r_{t+k}|\mathcal{F}_t] = \sum_{j=0}^{k-1} b_j \epsilon_{t+k-j}, \tag{1.13}$$

$$Var[r_{t+k}|\mathcal{F}_t] = E[(r_{t+k} - E(r_{t+k}|\mathcal{F}_t])^2|\mathcal{F}_t] = \sigma^2 \sum_{j=0}^{k-1} b_j^2. \tag{1.14}$$

If $k \to \infty$ then a conditional prediction error variance tends to the unconditional variance [18]. A conclusion is that assuming a linear process for the financial time series results in a variance independent of time, which contradicts the fact of time-dependent variance in the financial return time series.

1.2.3 Distribution of Return Series

To determine the distribution of high volatile return series such as stock return or stock price change is a cumbersome task and admits deep statistical analysis. In [7] it was investigated that there are numerous parametric models fitting the distribution of returns, e.g. normal inverse Gaussian distributions, stable distribution, Student distribution, hyperbolic distribution, exponentially truncated stable distributions.

Practice showed that return series often have distributions with fat tails and excess kurtosis. As an example that deviates from the normal distribution consider Figure 1.1 below. The figure shows a histogram of returns of NASDAQ stock index based on the closing prices at the end of the month in the time period from February 1971 until October 2019.

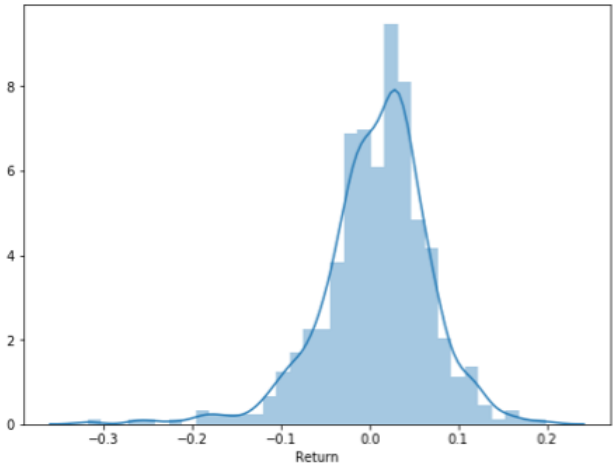

Figure 1.1: *Distribution of the return series of NASDAQ index from February 1971 until October 2019*

[18] states that recent researches showed that the mixture of the normal distributions is able to capture the complex structure of the return series.

1.3 Volatility forecasting

The volatility of asset returns is a measure of how much the return fluctuates around its mean and it is measured as the standard deviation of asset returns over a particular time period [15].

Various statistical models that are explaining the behavior of the variance and predicting future movements of financial time series have been developed. In Chapter 1.2.2 above, we have introduced a weak stationary process. Models suitable for this type of processes are the group of autoregressive moving average (ARMA) models. These models allow for future prediction of the returns in dependence on the weighted past return values. But, weak stationary processes assume a constant variance which contradicts the stylized facts of return time series. Thus variance modeling is crucial for making accurate forecasts of return series [18] and this is the main drawback of ARMA models. In the next chapter, we shall introduce a class of time series models that are able to overcome these shortcomings.

1.4 Time Series Models

One of the main stylized fact about the return series, introduced in Chapter 1.2.1, is heteroscedasticity. It describes the situation in which the variance of the asset return is not constant and is changing over the time. Figure 1.2 is showing how the variance of NASDAQ monthly returns is evolving over the time period from January 2000 until October 2019. The variance is calculated based on formula (3.3) which will be introduced in Chapter 3.

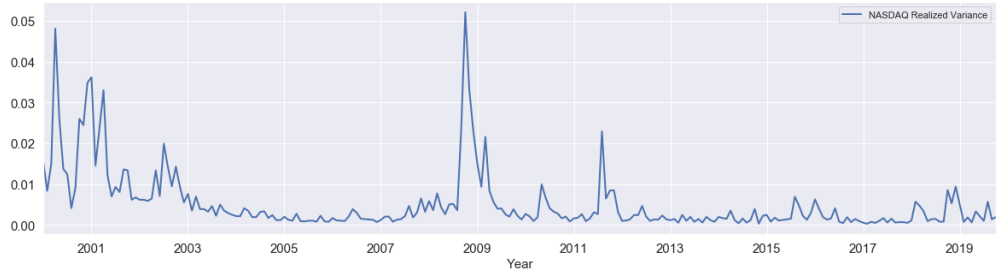

Figure 1.2: *Variance of NASDAQ monthly returns in the period from January 2000 until October 2019*

Time series models that assume a conditional variance which is changing over time, belong to the group of autoregressive conditional heteroscedastic (ARCH) models. ARCH models have been further improved which led to generalized ARCH (GARCH) models.

1.4.1 GARCH Model

As the name of generalized autoregressive conditional heteroscedastic model (GARCH) suggests, we deal here with a generalization of ARCH models. The ARCH model introduced by Engle [8] allows for a time dependent variance.

Let us considers the return process at the time t, $t \geq 0$

$$r_t = \mathbf{x_t}^\top \mathbf{b} + \epsilon_t, \tag{1.15}$$

r_t is a dependent variable, $\mathbf{x_t}^\top$ is a vector of exogenous variables (independent variables), $\mathbf{b}$ is the vector of unknown parameters, ϵ_t is an error term (return residual). Engel [8] proposes to use the previous information for ϵ_t. Hence,

ϵ_t is an adapted stochastic process which is a product of the i.i.d. random variables z_t and the time-dependent standard deviation σ_t such that

$$\epsilon_t = z_t \sigma_t \tag{1.16}$$

where z_t is a white noise, and

$$E(\epsilon_t | \mathcal{F}_{t-1}) = 0 \tag{1.17}$$

$$Var(\epsilon_t | \mathcal{F}_{t-1}) = \sigma_t^2. \tag{1.18}$$

Deterministic part of the process given in equation (1.15) is $\mathbf{x_t}^\top \mathbf{b}$, while ϵ_t is the random part which contains the conditional variance σ_t. The equation (1.15) can be written in the following form

$$\epsilon_t = r_t - \mathbf{x_t}^\top \mathbf{b}. \tag{1.19}$$

Assuming that ϵ_t is an adapted, normally distributed random process with mean 0 and variance σ_t^2

$$\epsilon_t | \mathcal{F}_{t-1} \sim N(0, \sigma_t^2), \tag{1.20}$$

the conditional variance of the return process r_t given by an ARCH (q) model is expressed as follows

$$\sigma_t^2 = \alpha_0 + \sum_{i=1}^{q} \alpha_i \epsilon_{t-i}^2. \tag{1.21}$$

To ensure a positive conditional variance $\alpha_0 > 0$ and $\alpha_i \geq 0$ for $i = 1, ..., q$.

The ARCH (q) model explicitly recognizes the difference between conditional and unconditional variance, allowing the conditional variance to change over time as a function of the past errors [5]. The generalization of the ARCH model is introduced by Borellslev in his paper from 1986 [5]. There, he states that in empirical applications of the ARCH model a relatively long lag in the conditional variance equation is often needed. To overcome the problem with negative variance parameter estimates, a fixed lag structure is imposed. This naturally requires an extension of the ARCH models to allow for a long memory and a more flexible lag structure.

Again, assuming a normal distribution of the adapted process ϵ_t

$$\epsilon_t | \mathcal{F}_{t-1} \sim N(0, \sigma_t^2), \tag{1.22}$$

the conditional variance of the return process r_t given by a GARCH (p, q) model is expressed as follows

$$\sigma_t^2 = \alpha_0 + \sum_{i=1}^{q} \alpha_i \epsilon_{t-i}^2 + \sum_{j=1}^{p} \beta_j \sigma_{t-j}^2 \tag{1.23}$$

where $p \geq 0$ and $q > 0$. To ensure that a variance is positive α and β are vectors of unknown parameters such that $\alpha_0 > 0$, $\alpha_i \geq 0$ where $i = 1, ..., q$, $\beta_j \geq 0$ and $j = 1, ..., p$.

If $p = 0$ the process reduces to the ARCH (q) process and if $p = q = 0$ the process is simply a white noise.

The difference between ARCH and GARCH models is that the variance modeled by ARCH is defined as a function of past sample variances only, wheres the GARCH model allows lagged conditional variances to enter as well [5].

1.4.2 Estimation of GARCH (p, q) Parameters

This chapter is based on the optimization method presented in [5].

Let us recall the return process at time t

$$r_t = \mathbf{x_t}^\top \mathbf{b} + \epsilon_t \tag{1.24}$$

where

$$\epsilon_t | \mathcal{F}_{t-1} \sim N(0, \sigma_t^2). \tag{1.25}$$

Let $\theta = (\mathbf{b}^\top, \alpha_0, \alpha_1, ..., \alpha_q, \beta_1, ..., \beta_p)$ be the vector of the parameters to be estimated. $\theta \in \Theta$ where Θ is a compact parameter space. To estimate the parameters of the GARCH (p, q) model, maximum likelihood (ML) algorithm can be used. Let $l_t(\theta)$ be the likelihood function of the observation at time t, then the maximum likelihood estimator for a sample of T observations with respect to the parameters in the vector θ is the following

$$\hat{\theta}_T = \arg\max_{\theta \in \Theta} \prod_{t=1}^{T} l_t(\theta). \tag{1.26}$$

Let us rewrite (1.24) in the following form

$$\epsilon_t = r_t - \mathbf{x_t}^\top \mathbf{b} \quad \text{and put} \quad \sigma_t^2 = \mathbf{z_t}^\top \omega \tag{1.27}$$

where $\mathbf{z_t}^\top = (1, \epsilon_{t-1}^2, ..., \epsilon_{t-q}^2, \sigma_{t-1}^2, ..., \sigma_{t-p}^2)$ and $\omega = (\alpha_0, \alpha_1, ..., \alpha_q, \beta_1, ..., \beta_p)$. Now, the vector of the parameters to be estimated becomes $\theta = (\mathbf{b}^\top, \omega^\top)$.

Assuming that ϵ_t is normally distributed with mean 0 and variance σ_t^2, the likelihood function at time t is derived from the probability density and it is given by

$$l_t(\theta) = \frac{1}{\sqrt{2\pi\sigma_t^2}} \exp^{-\frac{1}{2}\frac{\epsilon_t^2}{\sigma_t^2}}. \tag{1.28}$$

Instead, we will consider a logarithm of the likelihood function and it is calculated as follows

$$\begin{aligned} \log l_t(\theta) = \log l_t(\theta) &= \log(\frac{1}{\sqrt{2\pi\sigma_t^2}} \exp^{-\frac{1}{2}\frac{\epsilon^2}{\sigma_t^2}}) \\ &= \log \frac{1}{\sqrt{2\pi\sigma_t^2}} + \log \exp^{-\frac{1}{2}\frac{\epsilon_t^2}{\sigma_t^2}} \\ &= \log 1 - \log(2\pi\sigma_t^2)^{\frac{1}{2}} - \frac{1}{2}\frac{\epsilon_t^2}{\sigma_t^2} \\ &= 0 - \frac{1}{2}\log(2\pi) - \frac{1}{2}\log\sigma_t^2 - \frac{1}{2}\frac{\epsilon_t^2}{\sigma_t^2}, \end{aligned} \tag{1.29}$$

finally we have

$$\log l_t(\theta) = -\frac{1}{2}\log\sigma_t^2 - \frac{1}{2}\frac{\epsilon_t^2}{\sigma_t^2}. \tag{1.30}$$

The likelihood for a sample of T observations is now defined as the following

$$L_T(\theta) = \frac{1}{T}\sum_{t=1}^{T} \log l_t(\theta), \tag{1.31}$$

and the maximum likelihood estimator becomes

$$\hat{\theta}_T = \arg\max_{\theta\in\Theta} L_t(\theta). \tag{1.32}$$

An iterative gradient method is applied to obtain the set of the optimal GARCH (p, q) parameters. The gradient method is explained in [3]. Consider the gradient of $L_T(\theta)$, $\nabla_{L_T} = \frac{\partial L_T(\theta)}{\partial\theta}$, then any vector $\mathbf{d}$ in the same

halfspace as ∇_{L_T} (such that $\mathbf{d}^T \nabla_{L_T} > 0$) is a direction vector of increase of $L_T(\theta)$ such that $L_T(\theta + \lambda \mathbf{d})$ is an increasing function of the scalar λ.

A set of the directions $\mathbf{d}$ is given by

$$\mathbf{d} = \nabla_{L_T} H^{-1} \tag{1.33}$$

where H^{-1} is the inverse of the Hessian matrix (a second order partial derivative matrix of $L_T(\theta)$ with respect to the parameters θ). This method is known as the Quasi-Newton method. Computing the Hessian is costly and within the iterations, the matrix can become singular, hence not invertible. An alternative to the Hessian matrix is choosing a covariance matrix Q_T of the estimates which is necessarily positive definite, hence invertible.

The first order partial derivative of $\log l_t(\theta)$ with respect to the parameters in vector ω is given as follows

$$\begin{aligned}\frac{\partial \log l_t}{\partial \omega} &= -\frac{1}{2}\frac{1}{\sigma_t^2}\frac{\partial \sigma_t^2}{\partial \omega} - \frac{1}{2}\epsilon_t^2(-\frac{1}{\sigma_t^4})\frac{\partial \sigma_t^2}{\partial \omega} \\ &= \frac{1}{2}\frac{1}{\sigma_t^2}\frac{\partial \sigma_t^2}{\partial \omega}(\frac{\epsilon_t^2}{\sigma_t^2} - 1).\end{aligned}$$

The second order partial derivative is given by

$$\frac{\partial^2 \log l_t}{\partial \omega \partial \omega^\top} = (\frac{\epsilon_t^2}{\sigma_t^2} - 1)\frac{\partial}{\partial \omega^\top}(\frac{1}{2}\frac{1}{\sigma_t^2}\frac{\partial \sigma_t^2}{\partial \omega}) - \frac{1}{2}\frac{1}{\sigma_t^2}\frac{\partial \sigma_t^2}{\partial \omega}\frac{\partial \sigma_t^2}{\partial \omega^\top}\frac{\epsilon_t^2}{\sigma_t^2}$$

where

$$\frac{\partial \sigma_t^2}{\partial \omega} = z_t + \sum_{i=1}^{p} \beta_i \frac{\partial \sigma_{t-i}^2}{\partial \omega}. \tag{1.34}$$

The first order partial derivative of $\log l_t(\theta)$ with respect to the parameter vector $\mathbf{b}$ is the following

$$\begin{aligned}\frac{\partial \log l_t}{\partial \mathbf{b}} &= -\frac{1}{2}\frac{1}{\sigma_t^2}\frac{\partial \sigma_t^2}{\partial \mathbf{b}} - (\frac{1}{2}2\epsilon_t(-\mathbf{x_t})\frac{1}{\sigma_t^2} + \frac{1}{2}\epsilon_t^2(-\frac{1}{\sigma_t^4})\frac{\partial \sigma_t^2}{\partial \mathbf{b}} \\ &= -\frac{1}{2}\frac{1}{\sigma_t^2}\frac{\partial \sigma_t^2}{\partial \mathbf{b}} + \epsilon_t \mathbf{x_t}\frac{1}{\sigma_t^2} + \frac{1}{2}\frac{\epsilon_t^2}{\sigma_t^4}\frac{\partial \sigma_t^2}{\partial \mathbf{b}} \\ &= \epsilon_t \mathbf{x_t}\frac{1}{\sigma_t^2} + \frac{1}{2}\frac{1}{\sigma_t^2}\frac{\partial \sigma_t^2}{\partial \mathbf{b}}(\frac{\epsilon_t^2}{\sigma_t^2} - 1).\end{aligned}$$

The second order partial derivative is given by

$$\frac{\partial^2 \log l_t}{\partial \mathbf{b} \partial \mathbf{b}^\top} = -\frac{1}{\sigma_t^2}\mathbf{x_t}\mathbf{x_t}^\top - \frac{1}{2}\frac{1}{\sigma_t^4}\frac{\partial \sigma_t^2}{\partial \mathbf{b}}\frac{\partial \sigma_t^2}{\partial \mathbf{b}^\top}(\frac{\epsilon_t^2}{\sigma_t^2}) - 2\frac{1}{\sigma_t^4}\epsilon_t \mathbf{x_t}\frac{\partial \sigma_t^2}{\partial \mathbf{b}} + (\frac{\epsilon_t^2}{\sigma_t^2} - 1)\frac{\partial}{\partial \mathbf{b}^\top}(\frac{1}{2}\frac{1}{\sigma_t^2}\frac{\sigma_t^2}{\partial \mathbf{b}})$$

where

$$\frac{\partial \sigma_t^2}{\partial \mathbf{b}} = -2\sum_{i=1}^{q}\alpha_i \mathbf{x_{t-i}}\epsilon_{t-i} + \sum_{j=}^{p}\beta_j \frac{\partial \sigma_{t-j}^2}{\partial \mathbf{b}}. \tag{1.35}$$

Now, let $\theta^{(i)}$ be the i^{th} iteration of the parameter vector θ of $L_T(\theta)$, then the parameter vector in the next iteration is given by

$$\theta^{(i+1)} = \theta^{(i)} + \lambda_i Q_T^{-1}\sum_{t=1}^{\top}\frac{\partial \log l_t}{\partial \theta} \tag{1.36}$$

where λ_i is the scalar value at step i that maximizes $L_T(\theta)$ and ensures a convergence of the iterations to the point where the gradient is zero (for the criterion of choosing λ see [3]). λ is called learning rate and it will be described in Chapter 2. Q_T^{-1} is the covariance matrix of the gradient given by

$$Q_T^{-1} = E[\sum_{t=1}^{\top}\frac{\partial \log l_t}{\partial \theta}\frac{\partial \log l_t}{\partial \theta^\top}] \tag{1.37}$$

$$= E[\sum_{t=1}^{\top}\frac{\partial^2 \log l_t}{\partial \theta \partial \theta^\top}] \tag{1.38}$$

where $\frac{\partial \log l_t}{\partial \theta}$ is evaluated at $\theta^{(i)}$. The set of initial parameters θ can be randomly assigned.

Non-linearity of the likelihood function can lead to obtain a local maximum instead of a global one. The derivative of the likelihood function is 0 at both, local and global maxima and also at saddle points, thus the vector of the parameters θ does not necessarily lead to the global maximum of the likelihood. If $L_T(\theta)$ reaches a local maximum, the process will stop, as the gradient ∇_{L_T} is zero at this point. This problem is difficult to overcome. As a precaution, several initial values of θ can be chosen. If those initial values do not lead to the same convergence point, then the actual shape of the function should be investigated ensuring that the global maximum is located [3].

1.4.3 Exponential GARCH (EGARCH)

In Chapter 1.2.1 the leverage effect is introduced as a stylized fact of the returns. Exponential GARCH (EGARCH) models are formulated with a variance equation depending on the sign and size of the lagged residuals, whereas GARCH models take only the magnitudes of the lagged residuals and not their signs into consideration [18]. The advantage of the EGARCH models is its ability to capture the leverage effect. Consider equations given by (1.15) - (1.18), then the conditional variance given by EGARCH (p,q) model is the following

$$\ln(\sigma_t^2) = \alpha_0 + \sum_{j=1}^{p} \beta_j \ln(\sigma_{t-j}^2) + \sum_{i=1}^{q} (\alpha_i |\frac{\epsilon_{t-i}}{\sigma_{t-i}} - \sqrt{\frac{2}{\pi}}| + \gamma_i \frac{\epsilon_{t-i}}{\sigma_{t-i}}). \tag{1.39}$$

The leverage effect is described by the parameter γ such that $\gamma > 0$. The logarithmic function in equation (1.39) implies that the conditional variance given by an EGARCH (p,q) model is positive no matter what sign the residuals have.

There are many extensions of the GARCH models, but they will not be subject of this thesis.

Chapter 2

Artificial Neural Networks

2.1 Introduction to Artificial Neural Networks

Artificial neural networks (ANNs) are inspired by the learning process of the human brain [12]. They are constructed as a network of connected units called *nodes* or *neurons* where the connections are carrying certain weights which are giving an information on the importance of the neurons. The network has three basic sorts of layers: *input*, *hidden* and the *output* layer. The input layer is a vector containing values of given data observation (independent variable), the hidden layer represents the transformation of the input data and the output layer contains the result of the neural network (dependent variables). Figure 2.1 is showing a network consisting of one hidden layer that contains three neurons.

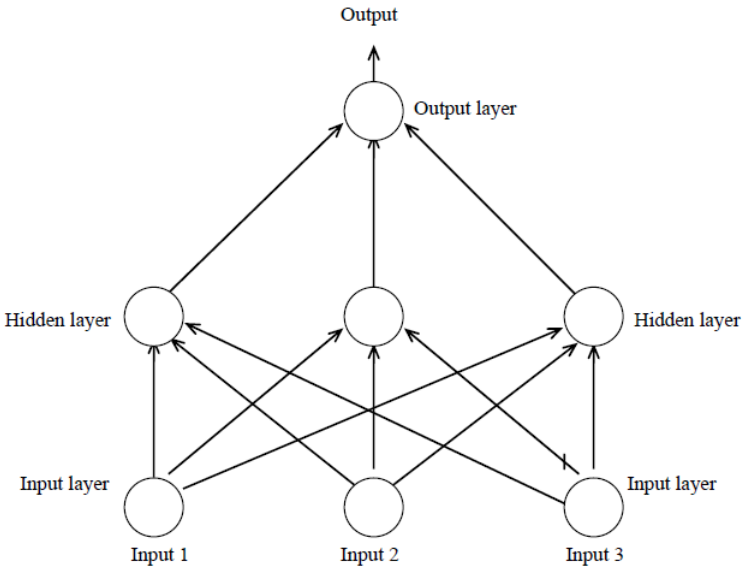

Figure 2.1: *Structure of the network with one hidden layer [11]*

The topology of the network is determined by the number of hidden layers, the number of the neurons in each layer and the nature of the activation function (the term will be explained later in the text) [12]. The number of hidden layers, as well as the number of neurons in each hidden layer, can be different and can influence the network's (model's) performance.

ANN is based on two processes, namely *forward propagation* and *back propagation*. For prediction modeling problems such is regression, in forward propagation, the input is taken and processed through multiple neurons in the hidden layers, retrieving the result in the output layer. The predicted result is compared to the actual one (called target output) and the error is calculated. The aim is to minimize the error, which is achieved in the back propagation step. The error is minimized "traveling" back to the neurons in the network and adjusting the weights of the neurons such that the error becomes as small as possible.

2.1.1 Structure of the Network

The simplest type of the neural network is a *feed-forward network*. In the feed-forward neural networks, all neurons of a particular layer are connected

to the neurons in the layer next to it [12]. In this network, the information moves in only one direction, from the input nodes, through the hidden layers to the output nodes, there are no cycles or loops in the network [22].

Let us start with introducing a term so-called *perceptron*. A perceptron can be seen as a binary classifier that produces output 0 or 1. The perceptron is a linear classifier or a linear function.

Let $\mathbf{x} = (x_1, x_2, ..., x_n)^\top$ be an input vector, $x_i \in \mathbb{R}$. To each element of the input vector $\mathbf{x}$ a certain weight is assigned $\mathbf{w} = (w_1, w_2, ..., w_n)$, $w_i \in \mathbb{R}$. The weights give information on the importance of the respective inputs. The weighted sum of the input $\mathbf{x}$ is

$$s = \sum_{i=1}^{n} x_i w_i. \tag{2.1}$$

The weighted sum is then compared to some threshold b, $b \in \mathbb{R}$ and the perceptron is given as binary function $f(s)$ such that

$$f(s) = 1 \quad \text{if} \quad s \geq b$$

$$f(s) = 0 \quad \text{if} \quad s < b$$

The above expressions can be written as follows

$$f(s) = 1 \quad \text{if} \quad s - b \geq 0$$

$$f(s) = 0 \quad \text{if} \quad s - b < 0$$

In this case, b is often called *bias*. The bias can be considered as a measure of how easy is for perceptron to achieve the output 1.

Making small changes in the weights of the neurons should result in small changes in the output and having the predicted result closer to the actual one. A problem with the perceptron, however, is that small changes in its weights can cause large changes in the output (e.g. from 0 to 1). Another issue is that adjusting the weights can change the behaviour of the whole network, meaning that the network can produce the desired output for certain data observations but make wrong predictions for all other observations.

Therefore, *activation functions*, also called *transfer functions*, were introduced. The argument of the activation function is a weighted sum of inputs plus a bias term. Having an activation function allows for adjustments in the weights in order to obtain better predictions. Another difference, comparing to the perceptron, is that the outputs are not only binary, but can take any continuous value. The most common activation functions are the following [18]:

- binary function (perceptron): $f : \mathbb{R} \to \{0, 1\}$

$$f(s) = \begin{cases} 1, & s + b \geq 0 \\ 0, & s + b < 0 \end{cases}$$

- bipolar function: $f : \mathbb{R} \to \{-1, 1\}$

$$f(s) = \begin{cases} 1, & s + b \geq 0 \\ -1, & s + b < 0 \end{cases}$$

- sigmoid function: $f : \mathbb{R} \to [0, 1]$, $c \in \mathbb{R}$

$$f(s) = \frac{1}{1 + e^{-cs}}$$

- tangent hyperbolic: $f : \mathbb{R} \to [-1, 1]$, $c \in \mathbb{R}$

$$f(s) = \tanh(\frac{cs}{2})$$

- ReLU (Rectified liner unit): $f : \mathbb{R} \to [0, \infty)$

$$f(s) = \begin{cases} 0, & s + b \leq 0 \\ s, & s + b > 0 \end{cases}$$

For non-linear prediction problems, activation functions such as sigmoid or tangent hyperbolic are often used.

Consider Figure 2.2 which shows the activation function of a perceptron [12]. This is a step function that takes as values either 0 or 1. This is a

discontinuous function and not differentiable, therefore it is not suitable for the gradient optimization method which will be introduced later in the thesis.

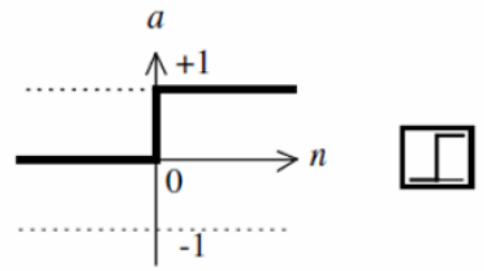

Figure 2.2: *Figure on the left represents the perceptron function. Figure on the right represents the symbol of the perceptron function [12]*

The sigmoid activation function is shown in Figure 2.3. This is a smooth nonlinear function that is differentiable and approximates well any value between 0 and 1 for the output [12].

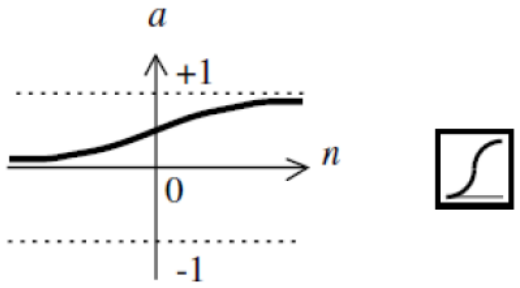

Figure 2.3: *Figure on the left represents the sigmoid function. Figure on the right represents the symbol of the sigmoid function [12]*

As already stated, the structure of the network can be complex, containing more than one hidden layer and a large number of neurons in each of them. This type of a neural network is called *multilayer neural network*. To determine the number of hidden layers in the network and the number of neurons in each layer, there is no universal approach or rule. For less complex tasks, one or two hidden layers can be used, but even for some highly complex problems a small number of hidden layers can result in a good performance of the network. These decisions depend on the specifications of the problem.

2.2 Estimation of the Network's Parameters

We will now consider a higher dimensional case of the input parameters. Hence, assume an input data set that contains independent variables. The input data set is a $n \times m$ matrix denoted by $\mathbf{X}$ whose entries are real numbers $x_{i,j}$ where i, j are elements of the index set I, and each column of the matrix represents one independent variable (one attribute),

$$\mathbf{X} = \begin{pmatrix} x_{1,1} & x_{1,2} & . & . & . & x_{1,m} \\ x_{2,1} & x_{2,2} & . & . & . & x_{2,m} \\ . & . & . & . & . & . \\ . & . & . & . & . & . \\ . & . & . & . & . & . \\ x_{n,1} & x_{n,2} & . & . & . & x_{n,m} \end{pmatrix}. \tag{2.2}$$

Each row of the matrix represents one observation of the data set. Each observation can be written in the form of the real valued vector $\mathbf{x}^d \in \mathbb{R}^m$ such that $d = 1, ..., n$.

A dependent variable is a vector of the length n denoted by $\mathbf{t}$ and it represents the desired output for each observation of the matrix $\mathbf{X}$ which is given by

$$\mathbf{t} = (t_1, t_2, ..., t_n)^\top. \tag{2.3}$$

The output vector $\mathbf{t}$ is called a *target* vector and $\mathbf{t} \in \mathbb{R}^n$.

The aims is to "predict" the values of the target vector $\mathbf{t}$ applying a neural network algorithm that is "trained" on the input data set $\mathbf{X}$.

Assume that the network has L layers, where $L \in \mathbb{N}^+/\{0\}$. Let $a^l_{d,j}$ be the output in the layer l of the neuron j with respect to the observation $\mathbf{x}^d$. Then the output in the first layer of the neuron j given the input vector $\mathbf{x}^d$ is given as

$$a^1_{d,j} = f(\sum_{i=1}^{m} w^1_{i,j} x_{d,i} + b^1_j) \tag{2.4}$$

where $w^1_{i,j}$ is the weight connecting i^{th} neuron from the input, in this case input $x_{d,i}$ to the j^{th} neuron in the first layer, $d = 1, ..., n$. Replacing an

argument of the activation function f with the following

$$o_{d,j}^1 = \sum_{i=1}^{m} w_{i,j}^1 x_{d,i} + b_j^1, \tag{2.5}$$

then the output $a_{d,j}^1$ becomes

$$a_{d,j}^1 = f(o_{d,j}^1), \quad d = 1, ..., n. \tag{2.6}$$

Analogously output in the layer $l+1$ of the neuron j given layer l is

$$a_{d,j}^{l+1} = f(o_{d,j}^l), \tag{2.7}$$

$d = 1, ..., n$, $l = 1, ..., L-1$ is the number of hidden layers and L is an output layer.

To estimate the performance of the network, an *error* function is calculated. The error is usually called *cost* or *loss* function and gives an information about how well the predicted output approximates the target output. In the empirical part of the thesis we implemented the mean squared error (MSE) and mean absolute error (MAE).

The MSE of the whole data set is calculated as an average of the errors for each observation d

$$E = \frac{1}{n} \sum_{d=1}^{n} E^d, \tag{2.8}$$

such that

$$E^d = \frac{1}{2}(t_d - a_d^L)^2 \tag{2.9}$$

where t_d is the target output and a_d^L is the output of the neural network in the last layer L.

The MAE of the whole data set is given as follows

$$E = \frac{1}{n} \sum_{d=1}^{n} E^d, \tag{2.10}$$

such that

$$E^d = \frac{1}{2}|t_d - a_d^L|. \tag{2.11}$$

After calculation of the error function, the algorithm "goes back" through the network to adjust the weights in order to minimize the error.

2.2.1 Gradient descent optimization

A standard method to obtain the optimal weights is the gradient descent algorithm in which network weights are moved in the direction in which the performance function (the error) is decreasing rapidly [12]. The gradient descent algorithm will give the new set of adjusted weights so that minimization of the error function is performed.

The gradient of the error function E with respect to each weight $w_{i,j}$ in each layer is calculated and $\Delta w_{i,j}$ is obtained for each $i, j \in I$. This shows how a change in that weight will affect the error function. Since the total error of the entire data set can be calculated as the sum of the errors in each observation

$$E = \sum_{d=1}^{n} E^d, \tag{2.12}$$

then the gradient of the entire data set can be calculated as the sum of the gradients in each data observation.

The gradient algorithm is introduced in Chapter 1.4.2 and it is based on the following: the initial set of the weights (or parameters) is assigned (usually randomly), then the iteration on the set of the parameters is performed by calculating first order derivatives with respect to the weights of the network until the error function is minimized. The new updated weight after k iterations is the weight

$$w_{i,j}^{k+1} = w_{i,j}^{k} + \eta \Delta w_{i,j}^{k} \tag{2.13}$$

where η is a *learning rate* which determines how much the weights are changed in each step.

There are several variants of the gradient descent and the most common ones are the following [21]:

i *batch gradient* - the gradient contributions for all data observations are accumulated before updating the weights. All observations of the training data set are passed through the network at once, the average loss is calculated and the weights are updated.

ii *stochastic gradient* - the weights are updated immediately after processing each data observation. One training observation is passed trough

the neural network at the time, loss is calculated and the weights in each layer are updated. The process is done for each training observation.

iii *mini-batches* - compromise between batch and stochastic gradient where the weights are updated after every n observations such that the training data set is partitioned into n parts (mini-batches).

ANNs are complex non-linear models, and while minimizing the error function the same problem that is occurring while maximizing the likelihood function for the statistical models, arises as well. This is the problem related to finding the local optimal solutions instead of the global ones. This problem is explained in [16]. Consider Figure 2.4 which illustrates the problem of globally optimal or globally minimal points for a non-linear function. A minimum of the function has a slope (derivative) equal to zero and a second derivative is positive, wheres a maximum of the function has a slope zero too, but the second derivative is negative. A saddle point has a slope and a second derivative both equal to zero. Figure 2.4 shows that the set of the initial weights may lie anywhere on the $x - axis$ close to the local or global maximum rather than the minimum. Updating and adjusting the weights, one can easily end up at one of the many positions where the derivative is zero and the curve has a flat slope [16].

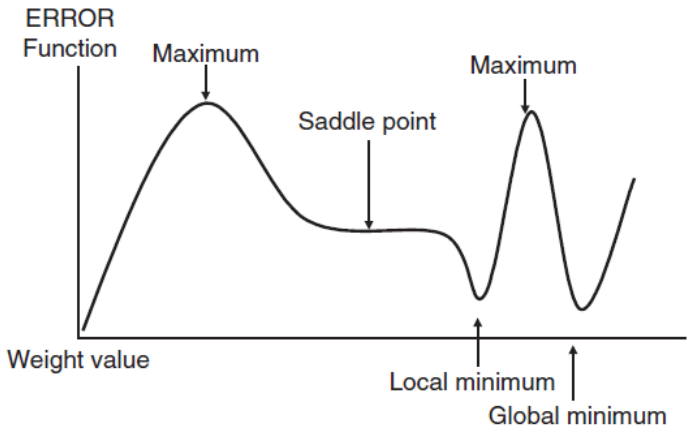

Figure 2.4: *Optimization problem with respect to the local minimum and maximum [16]*

2.2.2 Back Propagation Algorithm

The weights of the network are able to store the knowledge acquired during the learning process. A method that is used in learning processes is called learning algorithm and can be described as a function to adjust the weights to achieve the desired objective [21].

The most common learning algorithm is *back propagation* which is an iterative gradient based algorithm that minimizes an error between the target and the predicted output [21]. In this paper the back propagation which is based on batch gradient will be described.

Let us consider a generalized case, such that each observation in the input data set can have multiple outcomes. This leads to a target matrix instead of the target vector introduced in Chapter 2.2. Thus, the target matrix has the following form

$$\mathbf{t} = \begin{pmatrix} t_{1,1} & t_{1,2} & . & . & . & t_{1,k_L} \\ t_{2,1} & t_{2,2} & . & . & . & t_{2,k_L} \\ . & . & . & . & . & . \\ . & . & . & . & . & . \\ . & . & . & . & . & . \\ t_{n,1} & t_{n,2} & . & . & . & t_{n,k_L} \end{pmatrix}. \tag{2.14}$$

The Mean Squared Error function is then given by

$$E = \frac{1}{2n} \sum_{d=1}^{n} \sum_{j=1}^{k_L} (t_{d,j} - a_{d,j}^L)^2 \tag{2.15}$$

where $t_{d,j}$ is a target output with respect to the observation d, $a_{d,j}^L$ is an output of the neural network with respect to the observation d, and k_L is the number of outputs in the output layer L. The change in the weight in each layer l is a partial derivative of the error function with respect to that weight, i.e.

$$\Delta w_{i,j}^l = -\eta \frac{\partial E}{\partial w_{i,j}^l}. \tag{2.16}$$

The following back propagation algorithm is explained based on the calculation in [21].

The back propagation starts updating the weights in one to last layer $L-1$ and the change in the weight $w_{i,j}^{L-1}$ is then

$$\Delta w_{i,j}^{L-1} = -\frac{\eta}{2n} \sum_{d=1}^{n} \sum_{j=1}^{k_L} \frac{\partial}{\partial w_{i,j}^{L-1}} (t_{d,j} - a_{d,j}^{L})^2 \tag{2.17}$$

where k_L is the number of the neurons in the output layer. Since for the given j only the output $a_{d,j}^{L}$ has a relation to the weight $w_{i,j}^{L-1}$, it follows

$$\Delta w_{i,j}^{L-1} = \frac{\eta}{n} \sum_{d=1}^{n} (t_{d,j} - a_{d,j}^{L}) \frac{\partial a_{d,j}^{L}}{\partial w_{i,j}^{L-1}}. \tag{2.18}$$

Knowing that $a_{d,j}^{L} = f(o_{d,j}^{L-1})$, the chain rule is applied to calculate the partial derivative in the equation (2.18)

$$\frac{\partial a_{d,j}^{L}}{\partial w_{i,j}^{L-1}} = \frac{\partial a_{d,j}^{L}}{\partial o_{i,j}^{L-1}} \frac{o_{d,j}^{L-1}}{\partial w_{i,j}^{L-1}}. \tag{2.19}$$

The result is then

$$\frac{\partial a_{d,j}^{L}}{\partial o_{i,j}^{L-1}} = f'(o_{d,j}^{L-1}) \tag{2.20}$$

and

$$\frac{o_{d,j}^{L-1}}{\partial w_{i,j}^{L-1}} = \frac{\partial}{\partial w_{i,j}^{L-1}} \sum_{i=1}^{k_{L-1}} a_{d,i}^{L-1} w_{i,j}^{L-1} = a_{d,i}^{L-1} \tag{2.21}$$

where k_{L-1} is the number of the neurons in the layer $L-1$. From the equations (2.18) - (2.21) one can see the final result for the change of the weights $w_{i,j}^{L-1}$

$$\Delta w_{i,j}^{L-1} = \frac{\eta}{n} \sum_{d=1}^{n} (t_{d,j} - a_{d,j}^{L}) f'(o_{d,j}^{L-1}) a_{d,i}^{L-1} \tag{2.22}$$

$$= \frac{\eta}{n} \sum_{d=1}^{n} \delta_{d,j}^{L-1} a_{d,i}^{L-1} \tag{2.23}$$

where

$$\delta_{d,j}^{L-1} = (t_{d,j} - a_{d,j}^{L}) f'(o_{d,j}^{L-1}). \tag{2.24}$$

To make a generalization for the change of the weights in any layer l, one needs to calculate the change of the weights in $L-2$ layer too. That is the the following

$$\Delta w_{i,j}^{L-2} = -\eta \frac{\partial E}{\partial w_{i,j}^{L-2}}. \tag{2.25}$$

Let us recall that the weights from the layer $L-2$ are connected to the output j in layer $L-1$

$$o_{d,j}^{L-2} = \sum_{i=1}^{k_{L-2}} a_{d,i} w_{i,j}^{L-2} + b_j^{L-2} \tag{2.26}$$

and

$$a_{d,j}^{L-1} = f(o_{d,j}^{L-2}). \tag{2.27}$$

Hence there is the following

$$-\eta \frac{\partial E}{\partial w_{i,j}^{L-2}} = -\eta \sum_{d=1}^{n} \frac{\partial E}{\partial o_{d,j}^{L-2}} \frac{\partial o_{d,j}^{L-2}}{\partial w_{i,j}^{L-2}}. \tag{2.28}$$

From the previous calculation in (2.21) one knows that

$$\frac{\partial o_{d,j}^{L-2}}{\partial w_{i,j}^{L-2}} = a_{d,i}^{L-2}. \tag{2.29}$$

Further there is the following

$$\frac{\partial E}{\partial o_{d,j}^{L-2}} = \frac{\partial E}{\partial a_{d,j}^{L-1}} \frac{\partial a_{d,j}^{L-1}}{\partial o_{d,j}^{L-2}}. \tag{2.30}$$

Again from the previous calculation in (2.20) one knows that

$$\frac{\partial a_{d,j}^{L-1}}{\partial o_{d,j}^{L-2}} = f'(o_{d,j}^{L-2}). \tag{2.31}$$

Now, $\frac{\partial E}{\partial a_{d,j}^{L-1}}$ needs to be calculated. Since the algorithm goes backwards the following is obtained

$$\frac{\partial E}{\partial a_{d,j}^{L-1}} = \sum_{z=1}^{k_L} \frac{\partial E}{\partial o_{d,z}^{L-1}} \frac{\partial o_{d,z}^{L-1}}{\partial a_{d,j}^{L-1}} \tag{2.32}$$

$$= -\frac{1}{n} \sum_{z=1}^{k_L} \delta_{d,z}^{L-1} w_{j,z}^{L-1}. \tag{2.33}$$

From the equations (2.27)-(2.32) follows that the change of the weight in the layer $L-2$ is

$$\Delta w_{i,j} = \frac{\eta}{n} \sum_{d=1}^{n} (\sum_{z=1}^{k_L} \delta_{d,z}^{L-1} w_{j,z}^{L-1}) f'(o_{d,j}^{L-2}) a_{d,i}^{L-2}. \tag{2.34}$$

Writing

$$\delta_{d,j}^{L-2} = (\sum_{z=1}^{k_L} \delta_{d,z}^{L-1} w_{j,z}^{L-1}) f'(o_{d,j}^{L-2}), \tag{2.35}$$

the equation (2.34) becomes

$$\Delta w_{i,j}^{L-2} = \frac{\eta}{d} \sum_{d=1}^{n} \delta_{d,j}^{L-2} a_{d,i}^{L-2}. \tag{2.36}$$

Finally the generalization of the change of weights in any layer l, $l = 1, ..., L-1$ is obtained and it is given by

$$\Delta w_{i,j}^{l} = \frac{\eta}{n} \sum_{d=1}^{n} \delta_{d,j}^{l} a_{d,j}^{l} \tag{2.37}$$

where

$$\delta_{d,j}^{l} = (\sum_{z=1}^{k_{l+2}} \delta_{d,z}^{l+1} w_{j,z}^{l+1}) f'(o_{d,j}^{l}), \tag{2.38}$$

such that $l = 1, ..., L-2$.

The learning rate η has an important role. If η is too small then the algorithm will take a long time for the error to converge, on the other hand, if η is too large then the algorithm will end up bouncing around the error function, i.e. the algorithm diverges, see Figure 2.5.

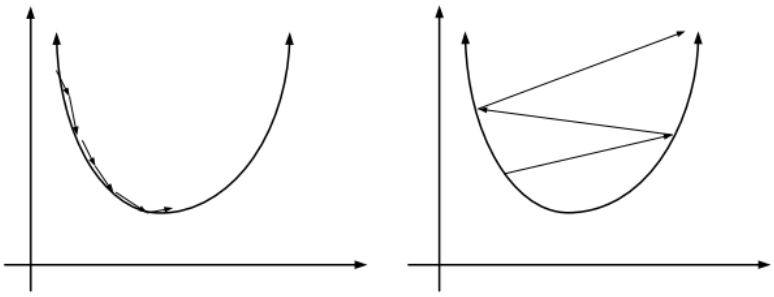

Figure 2.5: *Figure on the left represents a small learning rate which implies slow convergence. Figure on the right represents large learning rate which implies divergence [21]*

Chapter 3

Implementation of the Models

3.1 Data

In order to analyze the properties of returns and stylized facts of volatility on real data, we examined volatility of S&P 500 and NASDAQ stock indexes. We use S&P 500 data in the period of January 1970 until October 2019, while NASDAQ data is used from February 1971 until October 2019. The data consists of daily closing prices. S&P 500 contains 12571 and NASDAQ contains 12293 daily closing prices which are the basis for return and realized volatility calculation. The aim is to predict future values for the volatility based on historical data of returns and realized volatility.

An empirical part of the thesis is based on the implementation of two models, a neural network and GARCH models. The time frame taken for the prediction is one month, in average 20 trading days. In the neural network implementation, past observations of the monthly realized variance are used to predict the future, one month ahead variance. For GARCH models monthly returns are used in the variance prediction. The return at time t is calculated as a log return of the closing prices given by

$$r_t = \ln(\frac{P_t}{P_{t-1}}) \tag{3.1}$$

where P_t is the closing price at the time t.

Volatility is a latent variable and not directly observable [18]. Assuming

the return innovation is given by $r_t = \epsilon_t$ and $\epsilon_t = \sigma_t z_t$ where z_t is a white noise, $z_t \sim N(0,1)$, and it is independent of $\mathcal{F}_{t-1}$, then

$$E[r_t^2|\mathcal{F}_{t-1}] = E[\sigma_t^2 z_t^2|\mathcal{F}_{t-1}] = E[\sigma_t^2|\mathcal{F}_{t-1}] = \sigma_t^2 \tag{3.2}$$

and σ_t^2 is $\mathcal{F}_{t-1}$ measurable. Equation (3.2) appears to justify r_t^2 as a proxy for the true variance and it results in a very noisy measurement [18]. Relying on r_t^2 for the variance measure in terms of σ_t^2, realized monthly variance is then calculated as follows

$$RV_t = \sum_{m=1}^{n} r_{t+m}^2 \tag{3.3}$$

where

- n is the number of trading days in a month
- r_{t+m} is the return at the trading day m in the month t.

In the sequel we shall refer to *realized variance* because the calculation is based on the formula in equation (3.3).

After applying the formula given by (3.3), a time series contains 597 for S&P 500, and 584 observations for NASDAQ of monthly realized variances. The data sample in both cases is not large which is an additional challenge for the neural network. In case of both models, 20% of the data is set aside for the purpose of the out-of-sample testing and the rest, 80% represents a training data set. Out-of-sample results will be used to compare performance of the neural network and GARCH models.

3.1.1 Basic Statics about Data

Statics about monthly returns calculated based on the closing prices at the end of each month for S&P 500 and NASDAQ is given in Table 3.1.

Table 3.1: Basic statistics about S&P 500 and NASDAQ monthly returns

	S&P 500	NASDAQ
Number	597	584
Mean	0.00599	0.00754
Standard deviation	0.043608	0.060527
Min	-0.24542	-0.31792
Max	0.15104	0.19865
Kurtosis	2.67248	3.02367
Skew	-0.72354	-0.86932

In Figure 3.1 monthly returns of S&P 500 are shown.

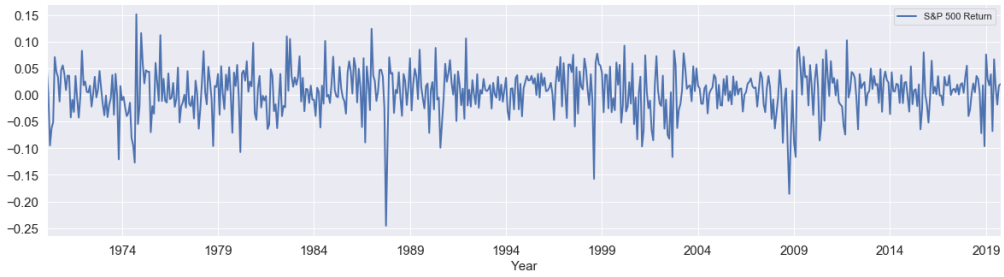

Figure 3.1: *Monthly returns of S&P 500 in the time period from January 1970 until October 2019*

NASDAQ monthly returns are shown in Figure 3.2.

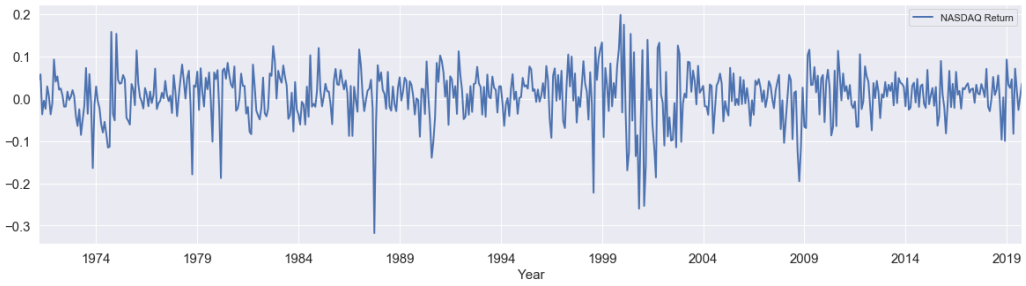

Figure 3.2: *Monthly returns of NASDAQ in the time period from February 1971 until October 2019*

A distribution function of S&P 500 monthly returns is shown in Figure 3.3.

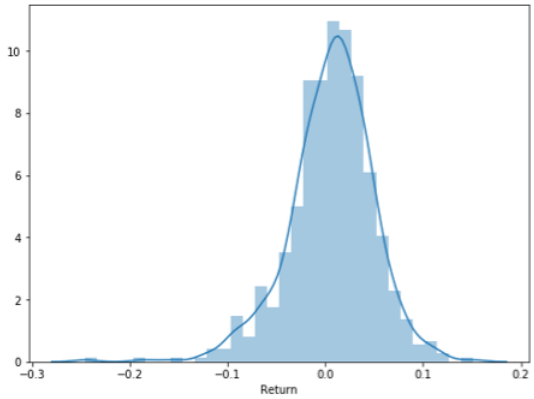

Figure 3.3: *Distribution function of S&P 500 monthly returns*

A distribution function of NASDAQ monthly returns is shown in Figure 3.4.

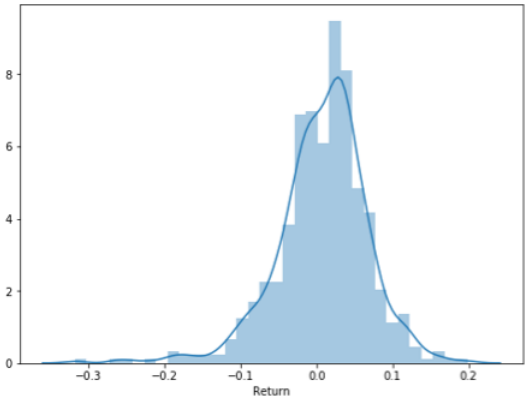

Figure 3.4: *Distribution function of NASDAQ monthly returns*

S&P 500 realized monthly variance calculated as in formula (3.3) can be seen in Figure 3.5.

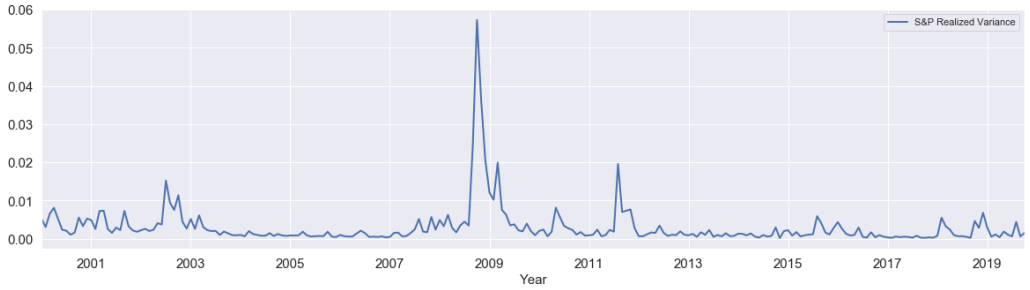

Figure 3.5: *Monthly realized variance for S&P 500 in period from January 2000 until October 2019*

NASDAQ realized monthly variance calculated as in formula (3.3) can be seen in Figure 3.6.

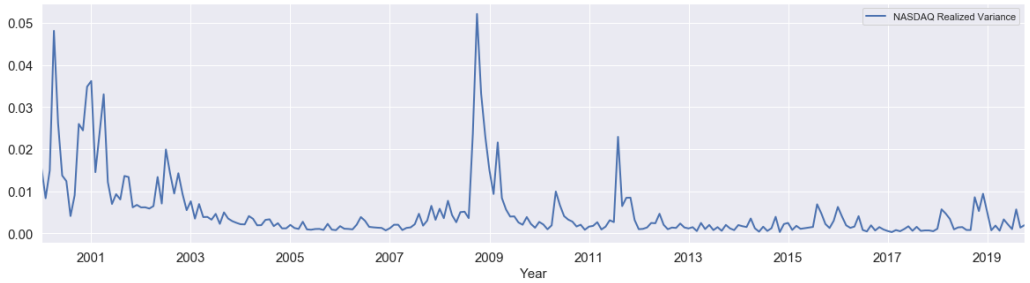

Figure 3.6: *Monthly realized variance for NASDAQ in period from January 2000 until October 2019*

From the Figures above, one can witness characteristics of the returns and realized variance as described in Chapter 1. Presence of the noise in the returns is observed from Figures 3.1 and 3.2, while the distribution that deviates from the normal can be seen in Figures 3.3 and 3.4. In the years 2008 and 2009, sudden jumps in the volatility development of S&P 500 and NASDAQ are observable (Figures 3.5 and 3.6) which is the result of the financial crisis in that period.

3.2 Results

Prediction of the future values for the variance in the case of neural network is based on the past observations of monthly realized variance calculated

as in equation (3.3), while the predicted variance with GARCH models is calculated by taking past observations of the monthly returns. Two metrics are implemented to evaluate the performance of the models. Those metrics are Mean Squared Error (MSE) given by (2.8) and Mean Absolute Value (MAE) given by (2.10). The results which are obtained by predicting a realized variance for the out-of-sample testing data, given by neural network and GARCH models, are compared.

3.2.1 Neural Network Results

There is no universal approach to determine the structure of the neural network suitable for the various problems, classification, prediction, clustering, etc. Hence, in order to get "the best" result of the network we used the "technique" of trials and error.

A time series consisting of the past monthly realized variances is transformed into the data set. We decided to create a data set that contains 6 variables (attributes/columns) out of which 5 variables are representing independent ones and the 6^{th} variable is dependent or target variable. This means that we would like to predict the variance on the basis of the last 6 observations. This decision comes from an experimentation with the different number of variables. It is also a consequence of the fact that the time series (consequently data set) is not large, and a large number of variables would increase complexity of the network which could lead to the wrong generalization and wrong prediction.

The data set is created as follows: the first observation (row) of the data set is obtained by taking a consecutive sequence of 6 observations (realized variances) from the series, staring at the first position. Then, the second observation (row) is obtained by taking a consecutive sequence of 6 observations (realized variances) from the series starting at the second position. This procedure is done until the series is exhausted.

The data set created for S&P 500 has 592 observations of monthly realized variances, 5 independent variables and 1 dependent variable (target variable). The data set created for NASDAQ has 579 observations of monthly realized variances, 5 independent variables and 1 dependent variable.

The predicted realized variance at the month t, RV_t is given by

$$RV_t = f(RV_{t-1}, RV_{t-2}, ..., RV_{t-5}) \tag{3.4}$$

where f is the function of the neural network.

A feed forward multiple layer network with a back propagation algorithm is implemented in the thesis for the prediction problem. We trained the network with 1 and 2 hidden layers because the larger number of hidden layers would result in the complex structure and would increase the running time. Hamid and Iqbal state in [11] that typically one hidden layer may be sufficient to map an input to the output in the financial forecasting. A number of neurons in each layer depends on the dimension of an input vector. An input vector has 5 variables, hence 5 neurons in the input layer. Too few neurons can affect the network to map incorrectly input into the output, while the large number of neurons can make the network to memorize the trivial patterns that can lead to the wrong generalization [11]. Our approach for choosing the number of neurons in hidden layer is the one proposed in [17], between the half the number and two times the number of input variables.

An activation function for the first hidden layer is sigmoid, described in Chapter 2.1.1. The second layer (if exists) has ReLU as an activation function (Chapter 2.1.1). ReLU activation function is chosen because it gives a positive output due to the fact that variance can not have negative values. A predicted result in the output layer is not processed through the activation function, it is a direct weighted input from the hidden layer.

Results of the neural network for the future S&P 500 realized variance prediction are given in Table 3.2. The data set for S&P 500 has 592 observations. The network is trained on 354 and validated on 119 samples (in-sample testing). Out-of-sample testing is done for the 20% of the data set, that is 119 samples taken as the last observations in the data set.

Results of the neural network for the future NASDAQ realized variance prediction are given in Table 3.3. The data set for NASDAQ has 579 observations. The network is trained on 347 and validated on 116 samples (in-sample testing). Out-of-sample testing is done for the 20% of the data set, that is 116 samples taken as the last observations in the data set. In both cases, the algorithm is taking around 25% of the training data for the in-sample testing

Table 3.2: Performance of the implemented neural network for S&P 500. Comparison of the metrics MSE and MAE for the out-of-sample observations. Column *No. iterations* is describing how many times the network is processing the training data and updating the weights, while the column *Batch size* is describing the size (number of observations) of the training data after which the weights are updated

No. hidden layers	No. neurons	MSE	MAE	No. iterations	Batch size
1	5	4.7730E-06	1.1793E-03	200	50
1	10	4.6103E-06	1.0927E-03	200	50
1	15	4.8886E-06	1.2809E-03	200	50
2	15-5	4.9760E-06	1.2646E-03	200	50
2	10-5	9.1985E-06	1.7508E-03	200	50
2	15-10	5.6017E-06	1.4130E-03	200	50

Table 3.3: Performance of the implemented neural network for NASDAQ. Comparison of the metrics MSE and MAE for the out-of-sample observations. Column *No. iterations* is describing how many times the network is processing the training data and updating the weights, while the column *Batch size* is describing the size (number of observations) of the training data after which the weights are updated

No. hidden layers	No. neurons	MSE	MAE	No. iterations	Batch size
1	5	1.1088E-05	2.0839E-03	200	50
1	10	9.6468E-06	1.8193E-03	200	50
1	15	7.2131E-06	1.4762E-03	200	50
2	10-5	7.8117E-06	1.3666E-03	200	50
2	15-10	8.0551E-06	1.6413E-03	200	50
2	20-10	7.4107E-06	1.5100E-03	200	50

(validation data).

A *k-fold* cross validation method is chosen to train the network. The number of splits (folds) is $k = 4$, which means that the algorithm is shuffling the training data sets, each time taking 3 set to train and the 4^{th} one to validated, Figure 3.7.

Figure 3.7: k-fold *cross validation method,* $k = 4$

Each step from Figure 3.7 is repeated a certain number of times. We choose 200 iterations for S&P 500 and NASDAQ, the larger number did not decrease the error.

3.2.2 GARCH Results

GARCH models are empirically proven to give the good results for the volatility (variance) prediction problems and in this thesis they are implemented in order to benchmark the result of the neural network.

GARCH models which are implemented in the thesis are: ARCH (1,1), GARCH (1,1) and EGARCH (1,1). These three models are implemented with a normal distribution and a Student-t distribution. Approximately 80% of the time series consisting of monthly returns is used to fit the models in order to estimate the parameters α_0, α and β. From Table 3.1 one can see that the mean of the monthly return time series is around zero, thus we decided to take the return process with a zero mean. Parameters are $p = 1$ and $q = 1$ for GARCH and EGARCH, and $q = 1$ for ARCH. Estimated parameters α_0, α, β and γ are then implemented in formulas (1.21), (1.23) and (1.39) and the future, one step ahead, conditional variance is predicted.

Predicted variance is then compared with the calculated realized variance as in formula (3.3), and MSE and MAE for the out-of-sample testing are calculated.

Table 3.4 is showing the performance of the GARCH models for S&P 500 tested on the out-of-sample data. Table 3.5 is showing the performance of the GARCH models for NASDAQ tested on the out-of-sample observations. In order to keep the consistency with the neural network out-of-sample results, we are producing the same number of the future predicted variances with GARCH models as with the neural network.

3.3 Discussion of the Results

In Table 3.6 "the best" results of the out-of-sample testing for neural network and GARCH models are presented for S&P 500 and NASDAQ prediction problems. "The best" results are based on the smallest value for Mean Squared Error and the smallest value for Mean Absolute Error.

Table 3.6: Comparison of the results obtained by neural network and GARCH models for the out-of-sample test data

Model	MSE	MAE
S&P 500 ANN (1 hidden layer, 10 neurons)	4.6103E-06	1.0927E-03
S&P 500 ANN (1 hidden layer, 15 neurons)	4.8886E-06	1.2809E-03
S&P 500 GARCH (1,1) (Student-t distribution)	4.9354E-06	1.2286E-03
S&P 500 EGARCH (1,1) (Student-t distribution)	4.9472E-06	1.2478E-03
NASDAQ ANN (1 hidden layer, 15 neurons)	7.2131E-06	1.4762E-03
NASDAQ ANN (2 hidden layer, 20 and 10 neurons)	7.4107E-06	1.5100E-03
NASDAQ GARCH (1,1) (Student-t distribution)	7.2740E-06	1.6886E-03
NASDAQ EGARCH (1,1) (Student-t distribution)	7.6420E-06	1.7748E-03

In the case of S&P 500 prediction problem, the smallest MSE and MAE are obtained by the neural network (1 hidden and 10 neurons). In the case of NASDAQ prediction problem, the neural network (1 hidden and 15 neurons) has the smallest MSE followed by the MSE of GARCH (1,1) with a Student-t distribution. The smallest MAE is obtained by neural network (1 hidden and

Table 3.4: Performance of the GARCH models for S&P 500. Presented results (MSE and MAE) are obtained from the out-of-sample testing for the future prediction of the realized variance. α_0, α and β are the estimated parameters. Models are fitted for the return time series containing 473 observations of the monthly returns, then 119 instances of the future, one step ahead, predicted variances are calculated and compared with the same number (approximately 20%) of the monthly realized variances given by formula (3.3). Akaike information criterion (AIC) and Bayesian information criterion (BIC) are presented too in order to compare the models.

Model	Distribution	MSE	MAE	α_0	α	β	AIC	BIC
GARCH (1,1)	Normal	4.9561E-06	1.2406E-03	9.0377E-05	0.1131	0.8522	-1613.47	-1600.97
GARCH (1,1)	Student t	4.9354E-06	1.2286E-03	1.3002E-04	0.1115	0.8302	-1633.25	-1616.59
EGARCH (1,1)	Normal	4.9641E-06	1.2574E-03	-3.5580E-01	0.2369	0.9418	-1616.15	-1603.66
EGARCH (1,1)	Student t	4.9472E-06	1.2478E-03	-4.4550E-01	0.2349	0.9277	-1633.86	-1617.19
ARCH (1,1)	Normal	5.0518E-06	1.3910E-03	1.8425E-03	0.1084		-1594.55	-1586.22
ARCH (1,1)	Student t	4.9744E-06	1.3738E-03	1.7969E-03	0.1308		-1617.57	-1605.08

Table 3.5: Performance of the GARCH models for NASDAQ. Presented results (MSE and MAE) are obtained from the out-of-sample testing for the future prediction of the realized variance. α_0, α and β are the estimated parameters. Models are fitted for the return time series containing 463 observations of the monthly returns, then 116 instances of the future, one step ahead, predicted variances are calculated and compared with the same number (approximately 20%) of the monthly realized variances given by formula (3.3). Akaike information criterion (AIC) and Bayesian information criterion (BIC) are presented too in order to compare the models.

Model	Distribution	MSE	MAE	α_0	α	β	AIC	BIC
GARCH (1,1)	Normal	7.6391E-06	1.9197E-03	4.1619E-04	0.1	0.8	-1288.98	-1276.54
GARCH (1,1)	Student t	7.2740E-06	1.6886E-03	2.6482E-04	0.1467	0.793	-1316.7	-1300.12
EGARCH (1,1)	Normal	7.7102E-06	1.8470E-03	-4.1010E-01	0.2388	0.9254	-1291.9	-1279.46
EGARCH (1,1)	Student t	7.6420E-06	1.7748E-03	-3.8410E-01	0.2786	0.9306	-1315.8	-1299.21
ARCH (1,1)	Normal	8.0705E-06	2.1849E-03	3.0926E-03	0.2692		-1258.3	-1250
ARCH (1,1)	Student t	7.7706E-06	2.1046E-03	2.8867E-03	0.3329		-1286.98	-1274.54

15 neurons). It is worth to mention that the implementation of the models is basic and the goal is to investigate how well the neural networks would perform when approximating a realized variance of the financial instruments. Based on the results in the thesis we cannot make a generalization and decide which model is performing better but, we can see that the neural networks are able to deal with a non-linear prediction problems and a noisy data. Advanced approach in the empirical part of the study would improve the results for both implemented models, neural networks and GARCH models.

Figure 3.8 is showing a plot of the target vector (calculated realized variance - formula (3.3)) against the neural network predicted result in the out-of-sample testing for S&P 500. The neural network consists of 1 hidden layer and 10 neurons. The predicted result obtained by the GARCH (1,1) model with a Student-t distribution can be seen in Figure 3.9.

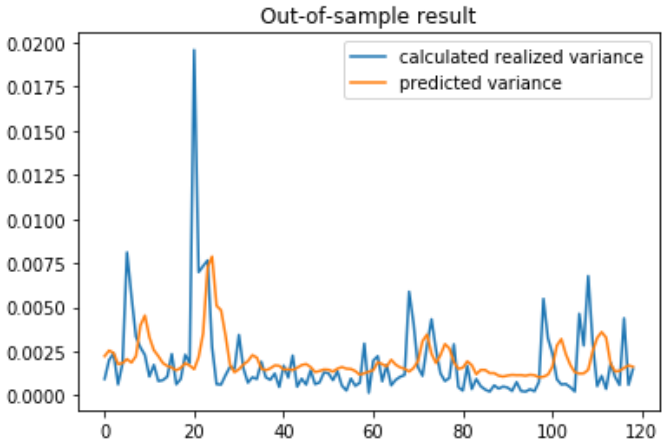

Figure 3.8: *Out-of-sample testing for S&P 500. Calculated realized variance (in blue) and predicted realized variance by neural network (in red). Neural network has 1 hidden layer and 10 neurons*

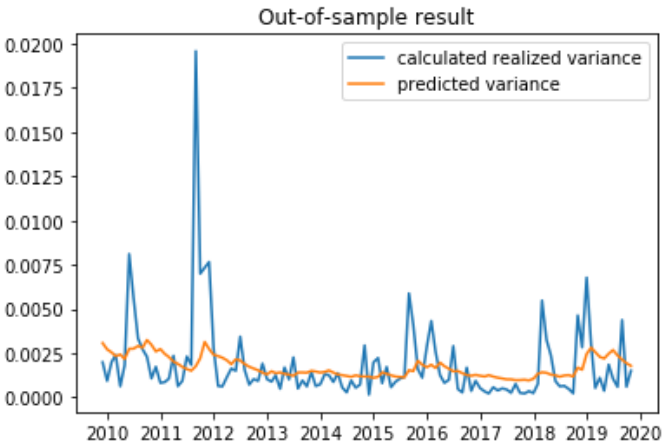

Figure 3.9: *Out-of-sample testing for S&P 500. Calculated realized variance (in blue) and predicted realized variance by GARCH (1,1) (in red). GARCH (1,1) model with a Student-t distribution*

Figure 3.10 is showing a plot of the target vector (calculated realized variance - formula (3.3)) against the neural network predicted result in the out-of-sample testing for NASDAQ. The neural network consists of 1 hidden layer and 15 neurons. The predicted result obtained by the GARCH (1,1) model with a Student-t distribution can be seen in Figure 3.11.

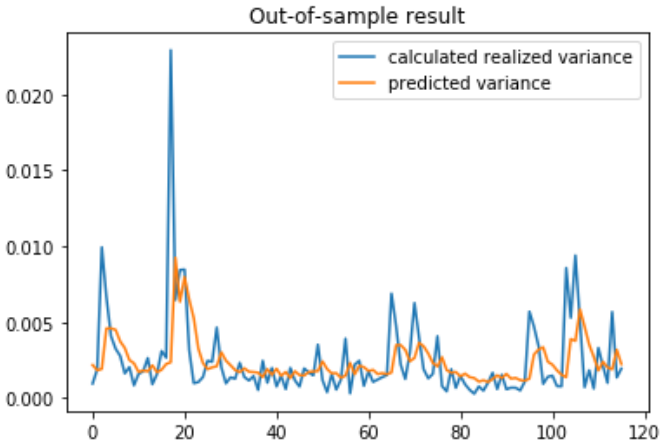

Figure 3.10: *Out-of-sample testing for NASDAQ. Calculated realized variance (in blue) and predicted realized variance by neural network (in red). Neural network has 1 hidden layer and 15 neurons*

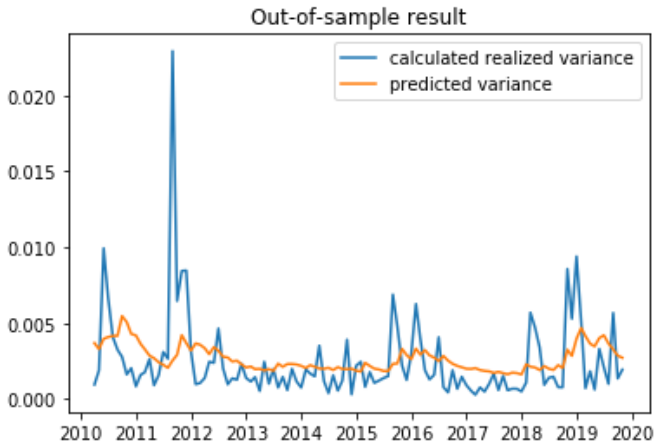

Figure 3.11: *Out-of-sample testing for NASDAQ. Calculated realized variance (in blue) and predicted realized variance by GARCH (1,1) (in red). GARCH (1,1) model with a Student-t distribution*

Plots of the in-sample testing are presented too. For the neural network in-sample testing we took a validation data set from the cross validation method. The validation data set is the end part of the whole data set used in the training procedure. The GARCH in-sample results are given by calculating the variance based on formulas (1.21), (1.23) and (1.39), implementing the estimated parameters and return time series that is used in the fitting procedure. In this case, we take the observations from the beginning of the series. Figures 3.12, 3.13, 3.14 and 3.15 are showing the plots of the calculated realized variance (formula (3.3)) against predicted variance obtained by neural network and GARCH models for S&P 500 and NASDAQ respectively.

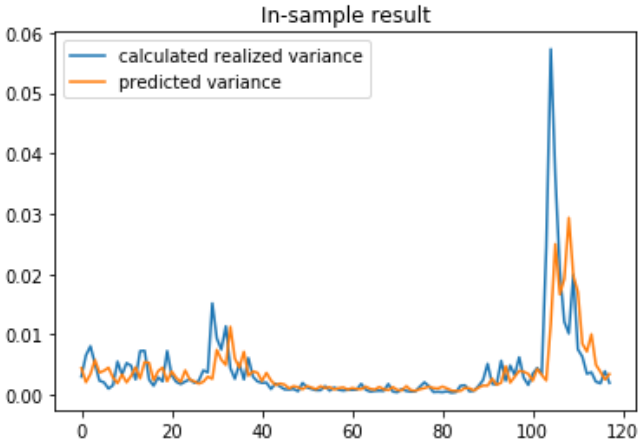

Figure 3.12: *In-sample testing for S&P 500. Calculated realized variance (in blue) and predicted realized variance (in red). Neural network has 1 hidden layer and 10 neurons.*

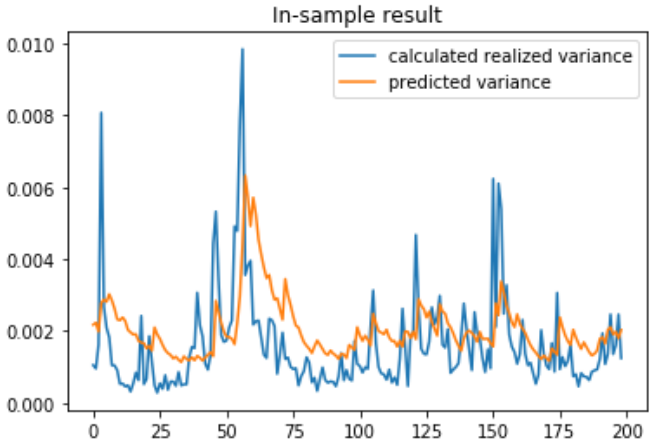

Figure 3.13: *In-sample testing for S&P 500. Calculated realized variance (in blue) and predicted realized variance (in red). GARCH (1,1) model with a Student-t distribution.*

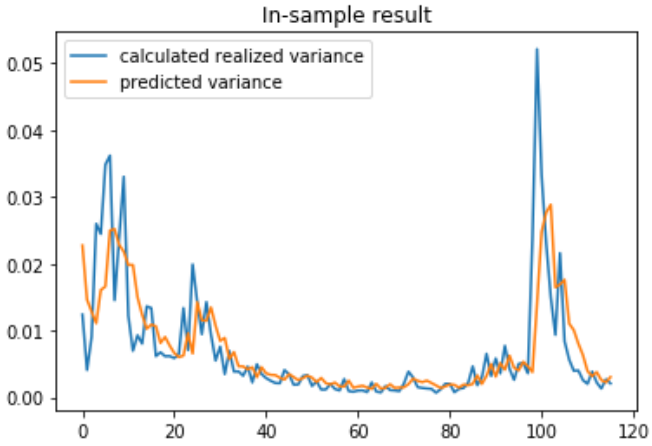

Figure 3.14: *In-sample testing for NASDAQ. Calculated realized variance (in blue) and predicted realized variance (in red). Neural network has 1 hidden layer and 15 neurons.*

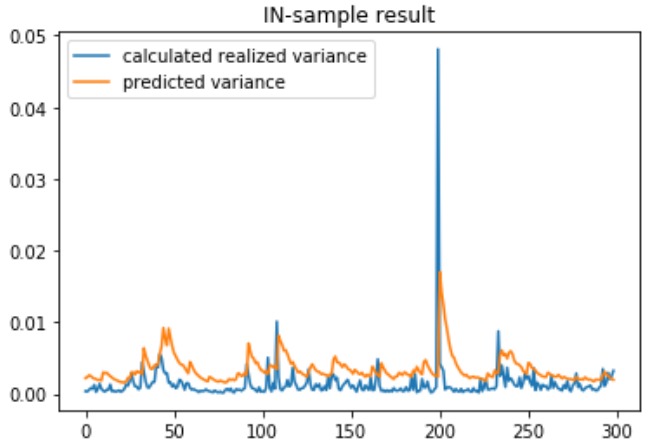

Figure 3.15: *In-sample testing for NASDAQ. Calculated realized variance (in blue) and predicted realized variance (in red). GARCH (1,1) model with a Student-t distribution.*

Conclusion

An objective of the thesis was to investigate how well the neural network algorithm would perform in the case of realized volatility forecasting of financial assets. An implementation of the neural network model is based on the simple feed forward network with a back propagation algorithm. The forecasting performance of the network is compared with the forecasting performance of the well known GARCH models. Given the results in the preceding chapter it is concluded that the neural network is performing similar to the GARCH models. It is important to state that the implementation of the neural network in the thesis is done having a very basic and simple model, and the training and testing data sets are not large. The simple feed forward network can be extended to the special form of the neural network called *recurrent network*. Recurrent networks have one or more feedback loops such that every loop can appear between any two neurons or layers, and it usually involves unit delay element [21]. These characteristics make recurrent neural networks advantageous in handling time series related problems over the feed forward networks [21].

Beyond the basic feed forward network, neural networks have been developed into more sophisticated and complex algorithms. The problem of the volatility forecasting is subject to many research papers where the scientists implemented advanced types of the networks which increased their performance. Authors in [1] and [4] implemented Radial Basis Function (RBF) neural networks where the results showed flexibility of RBF neural networks as a good modeling tool for the non-linear data. In [13] Hybrid neural networks-GARCH model are investigaded and it is shown that the neural networks can improve forecasting performance of the GARCH models for the given problem. [18] studied Mixture Density Networks (MDN) model, which is a combination of the neural networks and statistical model, mixture of den-

sity functions. The results of the MDN model outperformed GARCH models.

The advantage of the neural networks approach compared to the statistical models is that the networks are data-driven, no prior distribution assumption for the variables is needed, no necessary prior knowledge of the relationship between variables in the data and they are easily extended [18].

The results given by neural network for the volatility forecasting problems in the thesis can be significantly improved by choosing more suitable types of networks, increasing the complexity and tuning the parameters, and of course working with the larger training and testing data sets.

List of Figures

List of Tables

Bibliography

[1] O. Akbilgic, H. Bozdogan, and M. E. Balaban. A novel hybrid rbf neural networks model as a forecaster. *Statistics and Computing*, 24(3):365–375, 2014.

[2] L. Bauwens, C. M. Hafner, and S. Laurent. *Handbook of volatility models and their applications*, volume 3. John Wiley & Sons, 2012.

[3] E. R. Berndt, B. H. Hall, R. E. Hall, and J. A. Hausman. Estimation and inference in nonlinear structural models. In *Annals of Economic and Social Measurement, Volume 3, number 4*, pages 653–665. NBER, 1974.

[4] M. Bildirici and Ö. Ersin. Forecasting volatility in oil prices with a class of nonlinear volatility models: smooth transition rbf and mlp neural networks augmented garch approach. *Petroleum Science*, 12(3):534–552, 2015.

[5] T. Bollerslev. Generalized autoregressive conditional heteroskedasticity. *Journal of econometrics*, 31(3):307–327, 1986.

[6] D. R. Brillinger. *Time series: data analysis and theory*, volume 36. Siam, 1981.

[7] R. Cont. Empirical properties of asset returns: stylized facts and statistical issues. 2001.

[8] R. F. Engle. Autoregressive conditional heteroscedasticity with estimates of the variance of united kingdom inflation. *Econometrica: Journal of the Econometric Society*, pages 987–1007, 1982.

[9] I. Florescu. *Probability and stochastic processes*. John Wiley & Sons, 2014.

[10] W. A. Fuller. *Introduction to statistical time series*, volume 428. John Wiley & Sons, 2009.

[11] S. A. Hamid and Z. Iqbal. Using neural networks for forecasting volatility of s&p 500 index futures prices. *Journal of Business Research*, 57(10):1116–1125, 2004.

[12] A. R. Khataee and M. B. Kasiri. *Artificial neural network modeling of water and wastewater treatment processes*. Nova Science Publishers, 2011.

[13] W. Kristjanpoller, A. Fadic, and M. C. Minutolo. Volatility forecast using hybrid neural network models. *Expert Systems with Applications*, 41(5):2437–2442, 2014.

[14] B. B. Mandelbrot. The variation of certain speculative prices. In *Fractals and scaling in finance*, pages 371–418. Springer, 1997.

[15] S. Marra. Predicting volatility. *Lazard Asset Management*, 2015.

[16] P. D. McNelis. *Neural networks in finance: gaining predictive edge in the market*. Academic Press, 2005.

[17] L. Mendelsohn. Preprocessing data for neural networks. *Technical Analysis of Stocks and Commodities*, 11(10):416–420, 1993.

[18] F. Mostafa, T. Dillon, and E. Chang. *Computational intelligence applications to option pricing, volatility forecasting and value at risk*, volume 697. Springer, 2017.

[19] E. Osei-Wusu. *Relationship between Return, Volume and Volatility in the Ghana Stock Market*. PhD thesis, Thesis. Departmen of Financial and Statistics, Hanken School of Econometrics, 2011.

[20] S. E. Shreve. *Stochastic calculus for finance II: Continuous-time models*, volume 11. Springer Science & Business Media, 2004.

[21] C. T. Wai-shing and C. D. Siu-yeung. *Neural networks and computing: Learning algorithms and applications*, volume 7. World Scientific, 2007.

[22] A. Zell. *Simulation neuronaler netze*, volume 1. Addison-Wesley Bonn, 1994.