Diplomarbeit

The Impact of Market Sentiment and other Exogenous Variables on Corporate Bonds

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Finanz- und Versicherungsmathematik

eingereicht von

Dominik Springer
Matrikelnummer: 01255592

ausgeführt am Institut für Stochastik und Wirtschaftsmathematik
der Fakultät für Mathematik und Geoinformation der Technischen Universität Wien

Betreuung
Betreuer: Ao.Univ.Prof. Wolfgang Scherrer

Wien, 15.05.2018 _____ _____

(Unterschrift Verfasser) (Unterschrift Betreuer)

ABSTRACT

In this thesis we try to detect the possible impact of market sentiment and other exogenous variables, on corporate bond yields. After an Introduction that will analyse the available data, we will elaborate on some theory behind the techniques used to predict the future bond yields and the sentiment scores in general. We then talk about the general approach used for generating and comparing the forecasts. The final step will be to use four different approaches to try to predict the bond yields, based on their own history, sentiment scores and other explanatory variables. Starting with ARMA-GARCH and STAR models we will then take a look at recurrent neural network models. After introducing and using the Long short-term memory neural networks [12] we will apply the recently developed "Dual-Stage Attention-Based Recurrent Neural Network" [14] and measure its predictive power for the bond market.

"You do not study mathematics because it helps you build a bridge.
You study mathematics because it is the poetry of the universe.
Its beauty transcends mere things."

— Johnathan David Farley

ACKNOWLEDGMENTS

I want to thank all the people who accompanied me during my academic journey.
It was my girlfriend Johanna and her unconditional and warm-hearted support and my family's backing who made this thesis possible.

Special thanks go to Professor Wolfgang Scherrer for his thorough advice and guidance. His ongoing, extremely valuable supervision helped me develop knowledge in a domain I was completely new to.

CONTENTS

LIST OF FIGURES

Part I

DATA

1 INTRODUCTION

"I used to think that if there was reincarnation, I wanted to come back as the president, or the pope, or as a .400 baseball hitter. But now I would like to come back as the bond market. You can intimidate everybody.", James Carville, political advisor to President Clinton once said. This statement certainly can't be boldly used out of its context, but still shows the influence the bond market can have on markets and politics.
Often escaping broader media coverage, the bond market hides major predictive power, especially for real estate and equity markets. It is a great predictor of future economic activity and future inflation levels. All this correlations are clear and proven many times, but give rise to a question that hasn't been clearly answered yet.

Can we predict the bond market?

This question will be the main pillar this thesis is built upon.
The name bond, will be used as an acronym for corporate bond throughout this thesis. We define corporate bonds as all bonds excluding those, issued by governments in their own or any other currency, and those issued by supranational organisations (such as the European Bank for Reconstruction and Development (EBRD)).

The first chapter introduces the data available and tries to find exogenous variables that influence the bond prices. After examining the dependencies between those variables, the data will be checked for seasonality and stationarity.
Chapter 2 introduces some basic concepts that we will need in order to introduce our models later, that in turn will be used to predict the bond price movement.
After outlining the exact steps taken to predict those movements in Chapter 3, Chapter 4 introduces the 4 models used.
The first model will be a slight variation of the widely used ARMA-GARCH model, which allows for one external regressor. Secondly we will look at the smooth transition autoregressive (STAR) model, which again will yield a forecasts, which is a linear function of the past bond price movements, but will allow for different regimes. Those regimes will again depend on an external variable and could potentially model times of decisive events for bonds. In the section about available data we will also see a possible example of such a decisive event. The STAR and the ARMA-GARCH model, both delivering linear forecasts, should serve as a benchmark for the later introduced models belonging to the class of neural networks, that are capable of delivering non-linear forecasts. The comparison between the benchmark models and the two neural network models, should show, how much benefit can be drawn from the theoretical ability

to pick up non-linear dependencies over longer time horizons, or if these methods even yield inferior results.
After introducing the various models, we will start to choose the hyperparameters for the various models, that will yield the best forecasts. The hyperparameters will be chosen in three different ways, first we will determine them for every single bond, then we determine the hyperparameters that yield the best average result for a group of bonds belonging to the same company, and lastly we choose a single set of hyperparameters that yields the best average result for all bonds in our dataset. In Chapter 5 we then use these pa-rameter sets to conduct three predictions per model, one for every different approach of choosing our hyperparameters. This way we can theoretically account for company-wise patterns in the bond development with multiple parameter sets, or prevent over-fitting when using the single parameter set for all bonds.
Chapter 5 will also contain a section with results of the Diebold Mariano Test applied to our models pairwise and then will summarize the observed outcomes.
Chapter 6 finally puts the results into perspective, outlines other possible forecasting strategies and concludes the thesis.

DATASET: The historical sentiment data of the provider StockPulse used in this thesis, was made available free of charge by the company PS-Quant in the course of a research project conducted by them. As sentiment and bond data can in most cases only be obtained from contractors at a not negligible fee, I am very thankful for that contribution.
This data was complemented by other valuable information as seen later in this introduction. Nevertheless the following problems occurred:

- The time series length for the bond data was often too short to make a statistically reasonable forecast.
- The size of overlap of the different explanatory time series with the bond prices again was too short in many cases, further reducing the relevant data set

This leaves us with 91 datasets, with at least 800 entries for all of the variables. For a translation between the dataset numbers and the ISIN numbers, that uniquely identify the bonds, please consult Table A.1 of the Appendix.

SPREAD: In this thesis, our goal will be to predict the spread change of numerous bonds. To start ahead we will first build the basis to define that term.
The **yield** of a bond is the amount of money an investor will realize on one bond. As yield spread, or just spread, we will understand the difference between the yields of some bond, and the relatively risk-free U.S.Treasury

bonds' yield. At the current day t, the yield of a risk-free bond that matures in n days, is calculated via the Svensson-Model [11] as follows:

$$z_t^0(n) = \beta_t{}^0 + \beta_t{}^1 * \frac{1 - e^{-n/\tau_t^1}}{n/\tau_t^1} + \beta_t^2 * \left(\frac{1 - e^{-n/\tau_t^1}}{n/\tau_t^1} - e^{-n/\tau_t^1}\right) + \beta_t^3 * \left(\frac{1 - e^{-n/\tau_t^2}}{n/\tau_t^2} - e^{-n/\tau_t^2}\right)$$

With parameters $\beta_t^0, \beta_t^1, \beta_t^2, \beta_t^3, \tau_t^1$ and τ_t^2, that are estimated and published daily by the Federal Reserve (FED) for U.S. treasury bonds as used in this thesis, and the European Central bank for AAA-rated euro area central government bonds[1]. Let $z_t(n)$ be the yield, at the current day t, of some bond that matures in n days. The spread for this bond is $\Delta_t = (z_t(n) - z_t^0(n))$. The value of interest for us will be the spread change $y_t = \Delta_t - \Delta_{t-1}$ between two consecutive days. Spread usually does not stay the same, but widens and tightens, depending on a number of factors, including supply and demand, the risk associated with the emitting company and the overall state of the economy. Because yield embodies a relationship between the particular bond and a comparable treasury bond, it may get influenced by circumstances affecting the company, or the risk free treasury bond, or both at the same time.
Now that it is clear what we wan to predict, we want to introduce the set of explanatory variables available.

SENTIMENT , also market sentiment, is the general predominant view of investors on the future price development in a market. Contrarian investors believe that extreme market sentiment is a sign of impending market moves into the opposite direction of what sentiment predicted. But unfortunately things are not so clear-cut, as market moves also influence the sentiment. Of the four most popular ways to calculate sentiment listed below, we will use sentiment data generated by the 2^{nd} approach to complement our dataset.

- financial market data: There are several financial indicators that can be used to measure investor sentiment. A very popular approach is to look at the CBOE's Volatility Index (VIX), but there are many other possible factors to look at, such as the Price/Earnings Ratio, high yield bond returns or the TED spread (difference between the interest rates on interbank loans and on short-term U.S. government debt).

- textual data: Using information provided in newspapers, blogs, social networks or media platforms, is a obvious way to get sentiment information. What is less obvious is how this data is used to create sentiment scores. Counting certain positive and negative words, is a widely used approach, but recently also machine learning algorithms have been proposed to extract sentiment out of textual data.

1 The ecb's parameters can be accessed via https://www.ecb.europa.eu/stats/financial_markets_and_interest_rates/euro_area_yield_curves/html/index.en.html, the FED's via https://www.quandl.com/data/FED/PARAMS-US-Treasury-BETA-and-TAU-Parameters

- internet search behaviour: With the rise of search engines, there is plenty of data available, for example via Google Trends [2]. As with the previous example, there are several ways of using this data to generate a sentiment score, for example by determining words with negative and positive influence as done in [15].
- non-economic data: The last and least popular approach is to use non market related data to generate sentiment scores. The literature suggests using anything from dates of sports events to weather data in order to predict market moves.

We from now on will refer to the general form of textual sentiment as discussed above as sentiment. Another more specific form of sentiment, the buzz will be introduced later. Sentiment usually can be calculated using three different approaches. The "statistical approach", the "grammar approach" and the "machine learning approach". Below we want to give a rough and by no means complete overview on how those methods work.

1. For the **statistical approach**, news articles are scanned for words with positive and negative connotation. Those words are counted and given weights, which then yield the sentiment score
2. Most methods exploiting **grammar** in sentiment analysis make use of Hidden Markov Models, or Conditional Random fields.
3. The **deep learning model** creates a representation of entire sentences based on the sentence structure. It computes the sentiment taking into account the syntax of the phrases, not only the words used. This way, the model is more robust and not as easily fooled as other models. One could for example use an LSTM-network [12] to implement this idea as done in [7].

The sentiment data used in this thesis, was generated using the statistical approach.

BUZZ refers information created by the investor itself. The investor is emitter and receiver of the information. By expressing postures and opinions online, a social media buzz is generated. We will use the word buzz as the volume of that chatter throughout the internet.
Buzz can be seen as a proxy for investors attention and probably is as important as the message and thus the global sentiment or mood itself. More precisely, the buzz used here is a relative measure for the current volume of messages compared to the average exponentially smoothed volume, and therefore makes different titles comparable based on buzz.

STOCK PRICES tend to exhibit stronger volatility, and react stronger to news than bond prices. This might give us extra explanation power in our quest to predict the spread change. The stock prices are the closing prices

2 To be found at:https://trends.google.de/trends/

adjusted for stock splits and dividends, for each day, and were drawn from Yahoo Finance [3].

JOB MARKET DATA AND EARNINGS RELEASES: One of the most important events for the bond market, is the U.S. Labour Department's report. It includes information about all aspects of the job market, including unemployment rate, the number of jobs added or lost, average hourly wages and how the various sectors performed (manufacturing, etc.). The report is recurring on the first Friday of every month. Its importance cannot be understated. It is due to central position of the job market to the economy. To see two examples for the cross correlation of the spread change, that shows a very similar behavior for all datasets, see Figure 1.1 .
Another important factor for the bonds issued by a company, are the company's quarterly earnings reports (ER). They shed light on the current overall performance of the company, and include guidance for the next quarter. Both those events combined will be called news from now on.
Analysing the content and modelling the data provided by those reports, might give additional advantage. We however will settle for the date this releases occur, as it still shows some notable correlation to the spread change. Data for the news is drawn directly from the various bond emitting company's websites and from the "United States Department of Labour's homepage [4]. The news events enter the data as a column of flags, highlighting the relevant days, and as a column counting the days that have passed since the last news release.

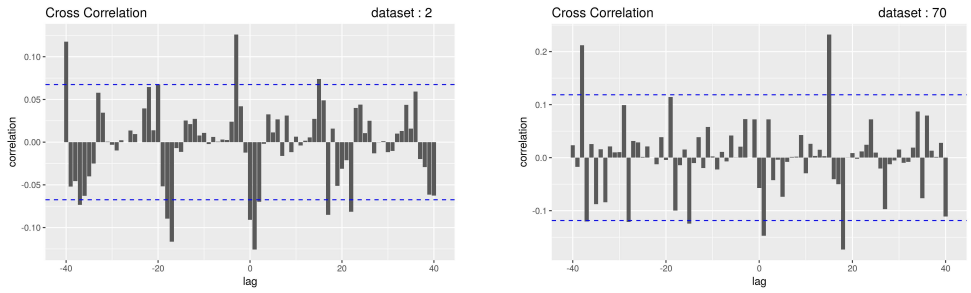

Figure 1.1: The cross correlation of the spread change and the days that passed since the job market report of bond 1(left) and bond 68(right)

TIME TO MATURITY indicates the time remaining until the end of the contract. This means this variable is declining every timestep until it finally hits 0.
As with the other datasets, we also tested time to maturity for correlation to the spread change. Both, the spearman's rho and the kendall's tau test, as implemented in *cor.test* of the core R package, did show p-values of under 0.05 for 45 of our 91 datasets. This suggests that the null hypothesis of no

3 To be found online at https://finance.yahoo.com/.
4 To be found online at https://www.bls.gov.

correlation might be untrue for those datasets.
Checking for correlations between mean-corrected values of the spread change and time to maturity, there are still 39 datasets with p-values of under 0.05. When taking the squares of those mean-corrected spread change values, the correlation to time to maturity gets even stronger, with now 82 datasets with small p-values. Interestingly, among the 9 bonds with relatively low correlation, there are all (3) bonds of the company "Societe Generale Group" that we have in our dataset, but no bonds of "Deutsche Bank AG", which constitute the biggest part of the dataset.

LAGGED SPREAD CHANGE is the original spread change series, shifted backwards in time. To identify dependencies of the lagged to the original series, we consider the autocorrelation function, that identifies correlations at various time lags. Figure 1.2 are typical examples of the acfs of our data samples. All of them exhibit significant correlation at a lag of approximately 19. Note that the news flag in our data sample recurs at a time lag of about 19 days. This is due to monthly recurring job market news, and the fact that days, for which not all of the data is available, are not included in our sample, making the average month in our samples about 19 days long.

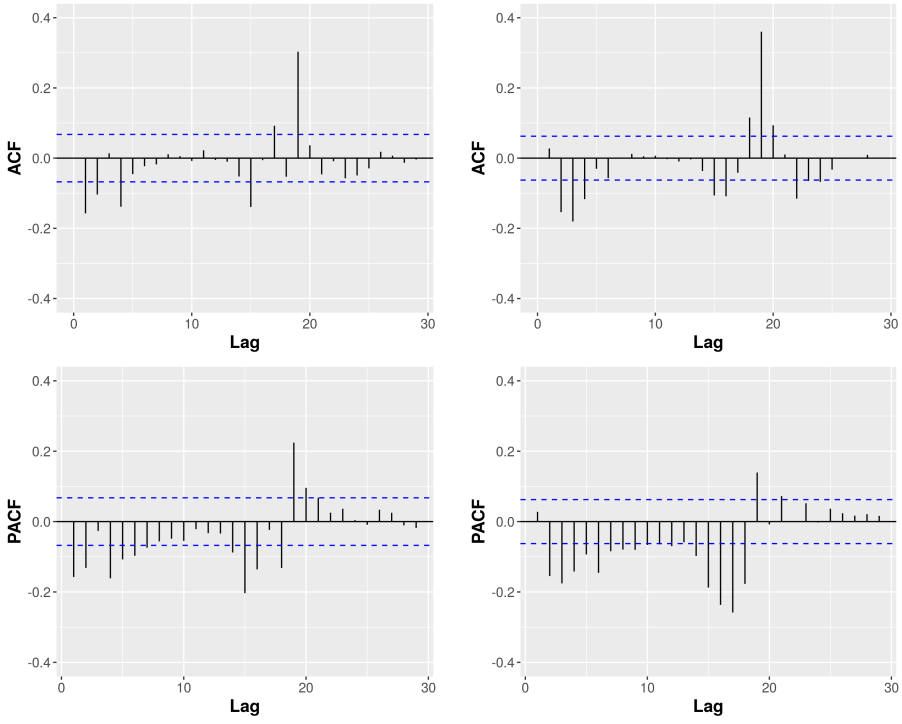

Figure 1.2: ACF and PACF of the spread change of dataset number 21(left) and 16(right)

RELATIONSHIPS: After having introduced numerous possible explanatory variables, we want to check their relationship to the spread change and the lagged spread change, as well as the relationships between each

other. The relationship to the one-day lagged spread change will also be of relevance, as that is the time horizon, that we will predict into the future. The interdependence of the exogenous variables can provide us of a better understanding of the data and identify redundant information.

From generated scatterplots, containing one explanatory variable and the lagged spread change at a time, it was obvious, that only one of the available variables, namely the news, had noticeable non-trivial significant impact on the spread change. Apart from that there was no observable pattern evident across different companies. The correlation between news and the spread change is visible in Figure 1.3. The only bonds that do not show this behaviour, are the bonds of data sets number 75 and 68.
Figure 1.3 shows that the spread change is more volatile shortly before and after news releases. Note that the second cluster of increased volatility, about 20 days after the news release, corresponds to the days before the next release.

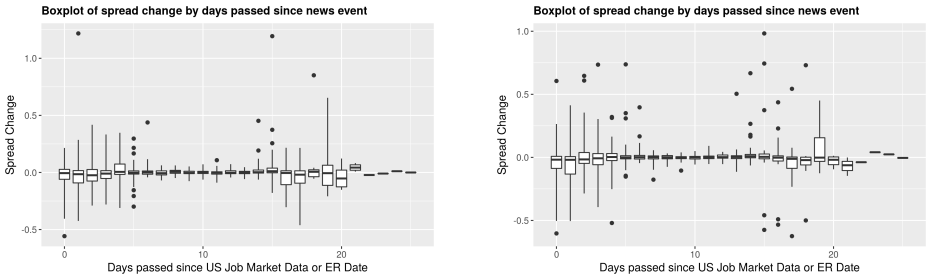

Figure 1.3: Boxplots for dataset nr. 4 and 13

Unsuprisingly there is also a correlation between the buzz and the sentiment. Looking at Figure 1.4 we observe that in periods of high buzz, the sentiment is more volatile. This can be intuitively explained by the fact, that strongly positive or strongly negative events tend to get more attention than neutral events, which we associate with a sentiment of around zero.

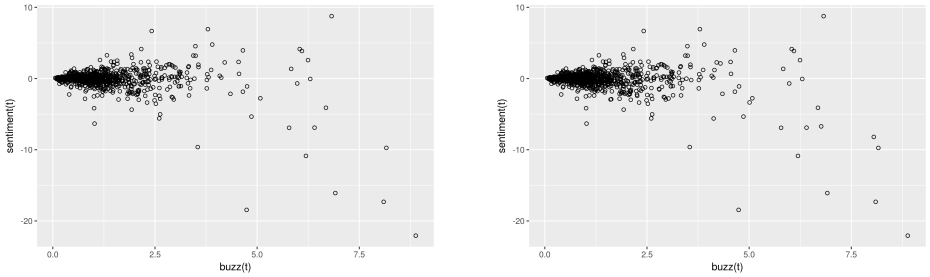

Figure 1.4: The scatterplot of buzz against sentiment for bonds 25 and 65

For bonds belonging to the same company the plots are obviously similar. This does accord with the observations made in the next paragraph. There we will see that the spreadchange of bonds of the same company behave similarly. This fact is illustrated in Figure 1.5

With the goal in mind, to predict the spread-change of our 91 bonds, we want to take a closer look at the spread change data generated.

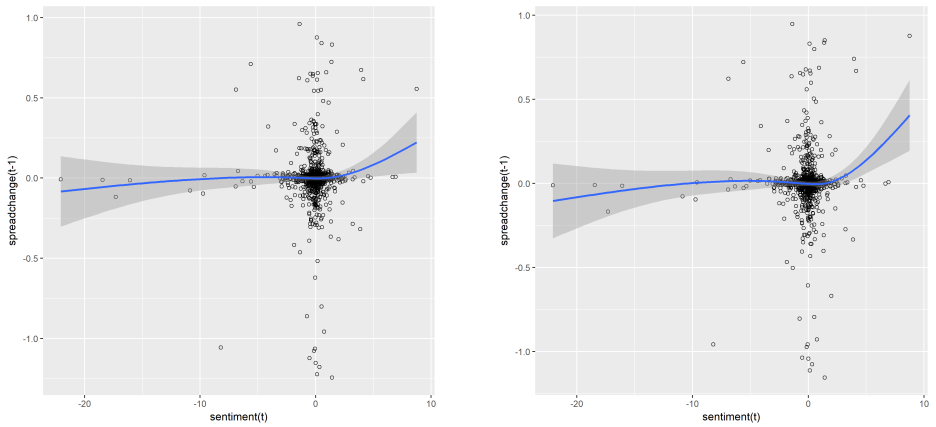

Figure 1.5: The scatterplot of sentiment against lagged spread change for two bonds for Deutsche Bank AG

SEASONALITY: As expected, the 91 time series exhibit no mean but in some cases some sort of seasonality as seen in Figure 1.6. What can be observed is, that bonds issued by the same company, show large volatilities at approximately the same times of the year, for several subsequent years. The direction of the price moves for those periods however seems to be ran-dom. Upon further inspection, most of those events can be identified as the cyclical earnings- and job-data-reports, described above, that occur at approximately the same time every month, respectively quarter.

STATIONARITY: After dealing with mean and seasonality we proceed with testing our null hypothesis of stationarity. We use a modified version of the test which was first introduced by Priestley-Subba Rao (PSR) in [9]. The modified PSR-test for stationarity is examining how homogeneous a set of spectral density function (SDF) estimates are across time and was adjusted by William Constantine to prevent bias of the estimators. The p-values of the test of stationarity are below 10^{-7} for all but 5 bonds in our dataset. Bonds of the data sets: 65, 68, 69, 70, and 75 still got negligibly small p-values of order 10^{-2} which lets us reject stationarity.

In Chapter 4 we will still try to model this time series via an ARMA-GARCH model, theoretically requiring stationarity. As a result we expect some correlation to be left when checking the residuals afterwards.

The above observations show that there is at best very little correlation between the spread change and any of the explaining variables. This is supported by the efficient market hypothesis, which states that stock prices are a function of information and rational expectations, and that newly revealed information about an asset's prospects is almost immediately priced-in. It would suggest, that the correlation between an asset's price change and relevant information appears with no, or almost no time-lag and can therefore not be used for predicting future price changes.
However we will further pursue our strategy, as even the slightest correlations, and the hopefully resulting advantage in the hit rate of our predic-

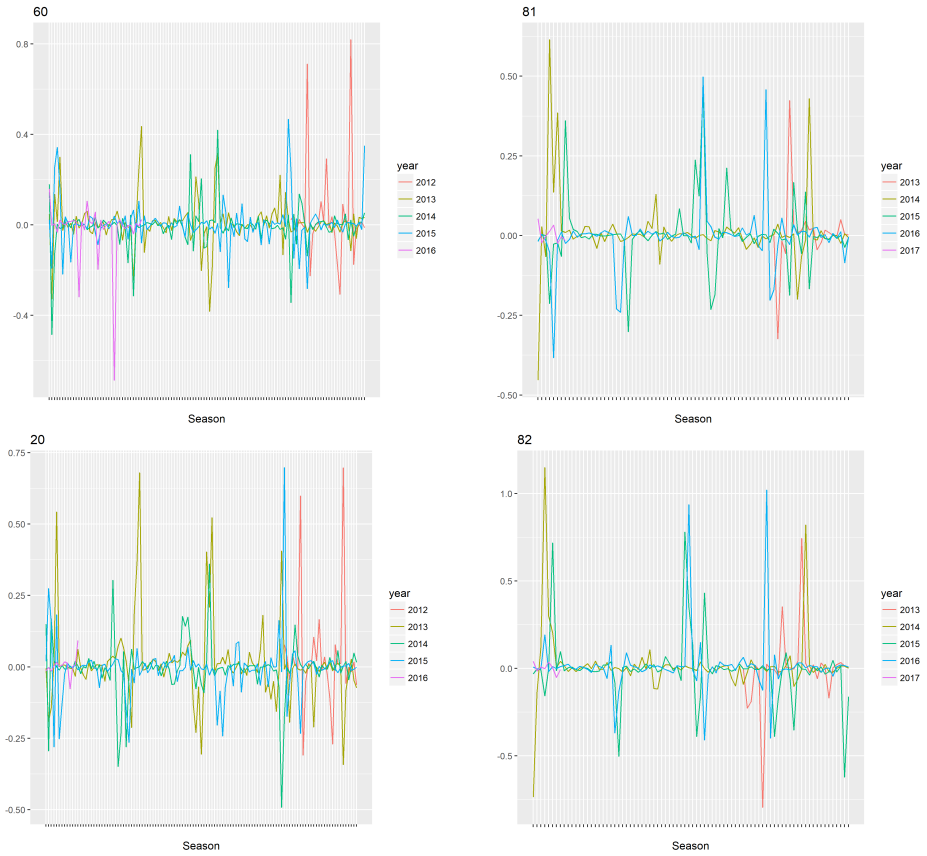

Figure 1.6: seasonal plots of the spread change of bonds issued by Deutsche Bank (left column) and by BNP Paribas (rigth column)

tions over random predictions, could be beneficially exploited in trading strategies. With random predictions we mean randomly choosing -1 or 1 as our predictions, achieving a hit rate of 0.5, as explained in Chapter 3. Furthermore using several explanatory variables at once to predict the spread change could yield satisfactory results, even when the correlations of the single variables to the spread change are negligible.

2

PRELIMINARY

Before starting with the theory of time series models, we want to cite some conceptcs, needed throughout this thesis.

Definition 2.1. *A process $(X_t)_{t\in\mathbb{Z}}$ is said to be (weakly)* ***stationary*** *[3] if*

- $\mathbb{E}(X_t^2) < \infty \quad \forall t \in \mathbb{Z}$
- $\mathbb{E}(X_t) = \mathbb{E}(X_s) \quad \forall t, s \in \mathbb{Z}$
- $Cov(X_t, X_s) = Cov(X_{t+k}, X_{s+k}) \quad \forall t, s, k \in \mathbb{Z}$

Definition 2.2. *Let $(X_t)_{t\in\mathbb{Z}}$ be a process and $F_{t_1,\dots,t_n}$ be the cumulative distribution function of the joint distribution of $(X_{t_1}, \dots, X_{t_n})$, then the process is called* ***strictly stationary*** *[3] if*

$$F_{t_1,\dots,t_n} = F_{t_1+k,\dots,t_n+k} \qquad \forall n \in \mathbb{N},\ t_i \in \mathbb{Z},\ k \in \mathbb{Z}$$

Definition 2.3. *A process $(\epsilon_t)_{t\in\mathbb{Z}}$ is called* ***white noise*** *[3] if it satisfies the following:*

- $\mathbb{E}(\epsilon_t) = 0 \qquad \forall t \in \mathbb{Z}$
- $Corr(\epsilon_t, \epsilon_s) = 0 \qquad \forall t, s \in \mathbb{Z},\ t \neq s$
- $\mathbb{V}(\epsilon_t) = \sigma^2 < \infty \qquad \forall t \in \mathbb{Z}$

it is called ***strong white noise*** *if*

- $(\epsilon_t)_{t\in\mathbb{Z}}$ is iid.
- $\mathbb{E}(\epsilon_t) = 0 \qquad \forall t \in \mathbb{Z}$
- $\mathbb{V}(\epsilon_t) = \sigma^2 < \infty \qquad \forall t \in \mathbb{Z}$

Note that white noise is stationary.

TIME SERIES MODELS In this paragraph we want to study several methods used to model the movement of time series. The probably simplest way to do so, is to model the current value x_t of the time series as being only dependent of its own past values.

Definition 2.4. *An* ***autoregressive(AR)*** *model of order p is defined as follows:*

$$X_t = c + \sum_{i=1}^{p} \phi_i X_{t-i} + \epsilon_t,$$

where $c, \phi_1, \dots, \phi_p$ are real valued parameters, $(\epsilon_t)_{t\in\mathbb{Z}}$ is white noise and $(X_t)_{t\in\mathbb{Z}}$ is the process of interest. [3].

Another similar concept is called the moving average model. It consist of an output variable that depends linearly on the current and various past values of a white noise process.

Definition 2 .5. *The **moving average (MA)** model of order q is defined as:*

$$X_t = c + \sum_{i=1}^{q} \theta_i \epsilon_{t-i},$$

where $(X_t)_{t\in\mathbb{Z}}$ is a process, $(\epsilon_t)_{t\in\mathbb{Z}}$ is a white noise process and the parameters of the model are the mean c and $\theta_1, \ldots, \theta_q$ [3].

The next model combines the above seen concepts. Autoregressive moving average (ARMA) models describe a process $(X_t)_{t\in\mathbb{Z}}$ via an autoregressive and a moving average part. With an optional external regressor added, we will call it an ARMAX model.

Definition 2.6. *An **ARMAX** model of order (p, q) with an external regressor is defined as:*

$$X_t = c + \sum_{i=1}^{p} \phi_i X_{t-i} + \epsilon_t + \sum_{i=1}^{q} \theta_i \epsilon_{t-i} + \zeta \nu_t$$

where $\phi_1, \ldots, \phi_p$ are the autoregressive parameters, $\theta_1, \ldots, \theta_q$ are the moving average parameters, c, ζ are constants, $\epsilon_t, \epsilon_{t-1}, \ldots$ are white noise error terms, ν_t is an optional external regressor and $(X_t)_{t\in\mathbb{Z}}$ the process to be modelled [3].

We will now define the autoregressive conditional heteroskedasticity(ARCH) model [3], a common model for the variance of a given time series:

Definition 2.7. *A time series $(X_t)_{t\in\mathbb{Z}}$ is called an **ARCH** series if:*

$$X_t = \sigma_t \epsilon_t$$

where $(\epsilon_t)_{t\in\mathbb{Z}}$ is a strong white noise and for σ_t we have:

$$\sigma_t^2 = \alpha_0 + \sum_{i=1}^{q} \alpha_i X_{t-i}^2$$

where $\alpha_0 > 0$ and $\alpha_i \geq 0, \;\; i > 0$ [3].

The generalisation of this model, the generalised autoregressive conditional heteroscedasticity (GARCH) model as found in [3], is defined as follows:

Definition 2.8. *A time series $(X_t)_{t\in\mathbb{Z}}$ is called **GARCH** series of order (p, q) if it follows:*

$$X_t = \sigma_t \epsilon_t$$

where $(\epsilon_t)_{t\in\mathbb{Z}}$ is a strong white noise process, and

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

where $\alpha_0, \ldots, \alpha_p, \;\; \beta_1, \ldots, \beta_q$ are non-negative real numbers with $\alpha_0 > 0$, $\alpha_p > 0$, and $\beta_q > 0$.

Before starting with another concept we cite an important result [3]

Theorem 2.1. *If we have*

$$\sum_{i=1}^{q} \alpha_i + \sum_{j=1}^{p} \beta_j < 1$$

then the unique strictly stationary process that satisfies the equalities of definition 2.8 is white noise.

2.1 NEURAL NETWORKS

We now want to cite some basic terms and definitions related to artificial neural networks. We however do not give the most general definition here, and refer the interested reader to [13]. A neural network (NN) is a collection of interconnected units called neurons. The output of each unit is a non-linear function of a weighted sum of the output of the other units. Suppose we have n units, then the output of the j-th unit is given by:

$$a_j = \phi_j(\sum_{i=1}^{n} w_{ji} a_i + b_j), \qquad j = 1, \dots n$$

where

- a_j is the output of neuron j, which is also called **activation**
- $\phi_j : \mathbb{R} \mapsto \mathbb{R}$ is the **activation function**
- $w_{ji} \in \mathbb{R}$ are some weights
- $b_j \in \mathbb{R}$ is called **threshold** or **bias**.

The interconnection structure of the net is determined by the weigths w_{ji}, $i, j = 1, 2 \dots, n$.
Input neurons are units without predecessors, i.e. neuron j is called an **input neuron**, if $w_{ji} = 0 \ \forall i$. Similarly output neurons are units without successors, i.e. neuron j is a **output neuron** if $w_{ij} = 0 \ \forall i$.
Input, respectively output neurons serve as an input-, output-interface for the whole network.

We say there is a **path** from neuron i to neuron j if there exist non-zero weights $(w_{k_1 k_0}, w_{k_2 k_1}, \dots, w_{k_u k_{u-1}})$ with $k_u = j$ and $k_0 = i$.

A **feedforward neural network** is a network without circles or loops, i.e. there does not exist a path from one neuron to itself. This implies that in a feedforward network information is only flowing in one direction.
A feedforward network defines a function which maps the activation of the input neurons to the activation of the output neurons. Suppose that the first n_i neurons are the input neurons and that the last n_0 neurons are the output

neurons and define $x = (a_1, a_2, \ldots, a_{n_i})$ and $\hat{y} = (a_{n-n_0+1}, a_{n-n_0+2}, \ldots, a_n)$ then the net gives a non-linear function:

$$\begin{aligned} f : \mathbb{R}^{n_i} &\mapsto \mathbb{R}^{n_0} \\ x &\mapsto \hat{y} \end{aligned}$$

Often one considers a feedforward NN which consists of so-called layers. The set of neurons $\{1, 2, \ldots, n\}$ is partitioned into subsets $I_k \subset \{1, 2, \ldots, n\}$, $k = 1, 2, \ldots, m$ called **layers**. The first layer I_1, called the **input layer**, contains the input neurons and the last layer I_m, called the **output layer**, contains the output neurons. The other $(m-2)$ layers are the so-called **hidden layers**. Furthermore there exist only connections from one layer to the following layer, i.e. if $i \in I_k$ then $w_{ji} = 0, \ \forall j \notin I_{k+1}$.

COMMON ACTIVATION FUNCTIONS The below listed functions are often used as activation functions in neural nets

Definition 2.9. *The **heaviside** function is given as:*

$$\phi(x) = \begin{cases} 1 & for \ x > 0 \\ 0 & for \ x \leq 0 \end{cases}$$

Note that for the neurons with a heaviside function as activation function the output is zero if the net input $(\sum_{i=1}^{n} w_{ji} a_i + b_j)$ is negative. This explains the term "threshold" for the parameter b_j. A single neuron with the heaviside function as activation function is called perceptron.
What follows is a function that we will use a lot throughout this thesis.

Definition 2.10. *The **hyperbolic tangent function** tanh is given as:*

$$tanh(x) = \frac{exp(x) - exp(-x)}{exp(x) + exp(-x)}$$

Another often used function is covered by the following definition.

Definition 2.11. *The **logistic function** f is defined as:*

$$f(x) = \frac{M}{1 + exp(-k(x - x_0))}$$

where $M \in \mathbb{R}, \ k \in \mathbb{R}, \ x_0 \in \mathbb{R}$.

Note that the function's graph is a sigmoid where: x_0 is the x-value of the sigmoid curve's midpoint, M is the curve's maximum and k is its steepness. Setting $L = 1, \ k = 1, \ x_0 \in \mathbb{R}$ we get:

Definition 2.12. *The **sigmoid function** S is defined by:*

$$S(x) = \frac{1}{1 + exp(-x)} = \frac{exp(x)}{exp(x) + 1}$$

A generalization of the logistic for multidimensional input is the softmax function. It however is no activation function!

Definition 2.13. *The **softmax function** is defined by:*

$$\sigma : \mathbb{R}^m \mapsto [0,1]^m$$
$$\sigma(x)_j = \frac{exp(z_j)}{\sum_{i=1}^{m} exp(z_i)} \qquad j = 1, \dots, m$$

We note that it takes as an argument a m-dimensional vector of real values and yields a m-dimensional vector of values between 0 and 1 that sum to 1. This function will be used to calculate weights later.

For the following discussion we set $w_{j0} = b_j$ and hence consider the thresholds as weights.
The aim of the network is to approximate a function f^0 by choosing optimal weights w_{ji}. Typically the function f^0 is unknown and one has only a sample of input, output pairs $(x_t, y_t)_{t=1,\dots,T}$. The goal then is to find weights such that the outputs $\hat{y}_t = f(x_t)$ of the net match the target values y_t as close as possible.
The network error $E(t)$ is a measure for the distance between the output $f(x_t)$ and the target y_t. Often the mean squared error is used, i.e. $E(t) = \frac{1}{n_0}(f(x_t) - y_t)^\top (f(x_t) - y_t)$
The estimation respectively the learning of the weights is often performed via an iterative schema called **learning rule**.

LEARNING RULE: One widely used class of algorithms to update the weights, the stochastic gradient descent (SGD), relies on the method of gradient descent by Newton. It is often used in combination with **Backpropagation** (BP), an efficient technique that computes gradients via the chain rule. For this method, we view the error E as a function of the weights w_{ij} and use gradient descent to minimize the error.
The weights are recursively updated as follows:

$$t = t + 1$$
if $t > T$
$$t = 1$$
end
$$w_{ji}^{(k+1)} = w_{ji}^{(k)} - \alpha \frac{\partial E(t)}{\partial w_{ji}}, \qquad j = 1, \dots n, \;\; i = 0, \dots, n$$

The above update is repeated until the weights converge or a prior given number of complete cycles through the sample $(1, \dots, T)$ have been performed. Such a cycle through the sample is called an **epoch**. Of course only weights w_{ji} are updated, that are not a priori set equal to zero. The design parameter α is also called the **learning rate**. Usually taking values below 0.1, the learning rate should be chosen small enough to ensure that the minima can be found, but big enough to ensure reasonably fast progress. As the learning algorithm proceeds, we will slowly adjust the learning rate, to help the algorithm to converge.

MINI BATCH: The process of presenting data to the neural network, checking the outcome by calculating the error and updating the weights of the network, is called supervised learning or simply **training**.
Using SGD for training the neural network models does give rise to two problems:

- Updating the weights each time data is presented to the network is computationally intensive
- The frequent updates can result in volatile model parameters which make it hard for the learning algorithm to find a minimum.

These problems are tackled by the so calles **mini batch stochastic gradient descent**. This algorithm collects a subset of input output pairs into a so-called **batch** with an a priori given **batch size**. Then the network error of this batch is computed as the sum over the network errors of the pairs in the batch. Next the weights are updated according to the gradient of the error of the batch.
This eliminates the first problem of the SGD, as the weights are updated far less often. The second problem also gets eliminated which results in smoother convergence. However the mini batch approach has its own downsides:

- The relatively low noise of the weight updates, compared to standard SGD, makes it more likely for the algorithm to land at a saddle point that it cannot "escape" from.
- Also due to the smoother updates, the learning algorithm may converge slower than with standard SGD.

Regarding to [2], faster convergence has been observed if the order in which the mini batches are visited is changed for each epoch. Later when training our models we will therefore shuffle our training sample before grouping it into batches. Our goal will be to find a good batch size that leaves us with the upsides of both those techniques while limiting the downsides and then choose a number of epochs that leaves us with reasonable computational time, and prevents "overfitting".

THE ADAM OPTIMISATION ALGORITHM: Apart from the mini batch SGD, we will also use a more recently developed algorithm, the adaptive moment estimation (**Adam**) optimizer [8]. It uses different adaptive learning rates for every weight in the neural network. Those learning rates are computed from estimates of the first and second moment of the gradient g_k at iteration step k.
Lets get more precise. The goal is to minimize

$$\mathbb{E}\left[E^{batch}(W)\right]$$

where W is a vector of the network's weights and $E^{batch} = \sum_{t \in batch} E(t)$ and *batch* is a randomly chosen subset of $\{1, \ldots, T\}$ of size T_b. Note that the stochasticity comes from the evaluation at random minibatches (subsamples) of datapoints.

The algorithm in every iteration updates the exponential moving average (EMA) of the gradient (m_k) and the EMA of the squared gradient (v_k), where hyperparameters $\beta_1, \beta_2 \in [0, 1)$ steer the exponential decay rates of the moving averages. These moving averages serve as estimates for the mean and the 2^{nd} moment of the gradient. The starting values of the two estimates however are chosen to be o-vectors which makes the estimates biased towards o. This effect will be countered by correcting the estimates, yielding the bias corrected estimates $\tilde{m}_k$ and $\tilde{v}_k$.

We will explain only the calculation of the corrected estimate for the second moment, as the calculation for the corrected first moment estimate follows the same steps. Note that we have:

$$\begin{aligned}
\mathbb{E}(v_k) &= \mathbb{E}(\beta_2 v_{k-1} + (1-\beta_2) g_k^2) \\
&= \mathbb{E}\left(\beta_2(\beta_2 v_{k-2} + (1-\beta_2) g_{k-1}^2) + (1-\beta_2) g_k^2\right) \\
&= \ldots \\
&= \mathbb{E}\left((1-\beta_2) \sum_{i=0}^{k-1} \beta_2^i g_{k-i}^2\right) \\
&= \mathbb{E}(g_k^2)(1-\beta_2) \sum_{i=0}^{k-1} \beta_2^i + \zeta \\
&= \mathbb{E}(g_k^2)(1-\beta_2) \frac{1-\beta_2^k}{1-\beta_2} \\
&= \mathbb{E}(g_k^2)(1-\beta_2^k) + \zeta
\end{aligned}$$

where $\zeta = 0$ if the true second moment $\mathbb{E}(g_{k-i}^2)$ is constant. If not, ζ can still be kept small since we can choose β_2 such that the EMA assigns small weights to gradients that lie too far in the past. For β_2 however, the paper suggests picking relatively high values in order to achieve fast convergence. From the above equations we conclude that the corrected estimate for the moving average of the 2^{nd} moment is $\tilde{v}_k = v_k(1-\beta_2^k)$. Remember that we have to correct the estimate because we initialize the running average with zeros.

Algorithm 1 does shortly outline the procedure to update our weights. Note that all operations are vectorwise.

The authors of the paper proposed the following choices for the parameters, which we will adopt for this thesis:

$\alpha = 0.001,\ \beta_1 = 0.9,\ \beta_2 = 0.999,\ \epsilon = 10^{-8}$.

Algorithm 1 : The Adam algorithm

Data :

- α (stepsize)
- $\beta_1, \beta_2 \in [0, 1)$ (exponential decay parameters)
- objective function $E^{batch}(W)$, with parameter vector W
- initial vector of network weights W_0

Result : updated network weights $W^{(k)}$
$m_0 \leftarrow 0$ (initialize 1^{st} moment vector)
$v_0 \leftarrow 0$ (initialize 2^{nd} moment vector)
while $W^{(k)}$ *not converged* **do**
  $k \leftarrow k + 1$
  $g_k \leftarrow \nabla_W E^{batch}(W^{(k-1)})$
  $m_k \leftarrow \beta_1 m_{k-1} + (1 - \beta_1) g_k$
  $v_k \leftarrow \beta_1 v_{k-1} + (1 - \beta_2) g_k^2$
  $\tilde{m}_k \leftarrow m_k / (1 - \beta_1^k)$ (bias-corrected first moment estimate)
  $\tilde{v}_k \leftarrow v_k / (1 - \beta_2^k)$ (bias-corrected second moment estimate)
  $W^{(k)} \leftarrow W^{(k-1)} - \alpha \tilde{m}_k / \sqrt{\tilde{v}_k} + \epsilon$
end
return $W^{(k)}$

In this thesis the task is to predict the spread change given the information up to a certain time. We consider two time series $y_1, \ldots, y_T$ and $x_1, \ldots, x_T$ where y_t are the spread changes and x_t is the time series which contains our exogenous variables.
The aim is to predict the value y_t of the output series at time t, given present and past values of the exogenous variables $x_t, x_{t-1}, \ldots, x_{t-P+1}$, where the number P is called **time horizon**. To this end we use specially structured NNs which are designed to "respect" the time series nature of the data. The network is split into P identical subnets. The activations of the previous subnet together with x_{t-P+k} is the input to the k-th subnet. The prediction for y_t then is (part of) the combined outputs of the generated subnets, or the output of the last subnet, as depicted in the following picture:

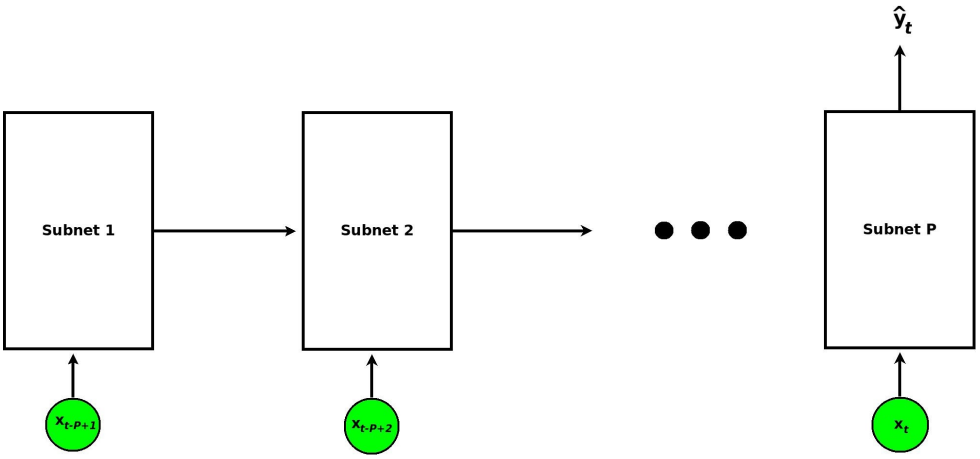

Figure 2.1: Subnets of a network

The subnets are identical, i.e. they have the same network structure and the same weights. Therefore such nets can also be represented by a recurrent net, where the outputs are fed back as inputs with a (time) delay:

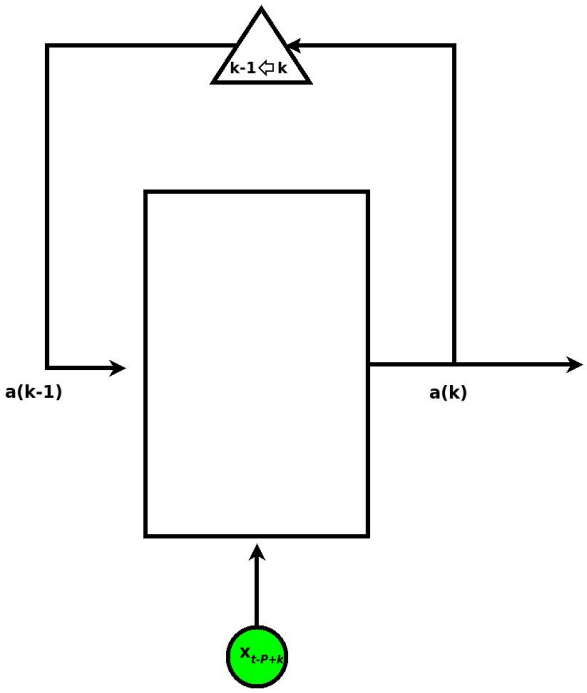

Figure 2.2: A recurrent unit

In Chapter 4 we will consider two ANNs of this kind, the LSTM network and the DA-RNN network.

Part II

THE MODELS

3

THE APPROACH

3.1 OVERVIEW

When thinking of forecasting time series, the first thing that naturally comes to ones mind, are ARMA models. An autoregressive moving average model of order (P, Q) as seen in Chapter 2 is easy to apply as theory and practical implementations are readily at hand. Two of the methods we will use for prediction are non-linear models, one is based on the ARMA, one on the AR model, but both allow for external regressors. The other two models we will be using, belong to the class of artificial neural networks. By comparing the results generated by those 4 networks, we want to see and quantify the advantage we get, by using more complex models, and to what extent any of those models can reach our goal, namely to predict the movements of the spread change correctly.

3.2 RANDOMIZING

For the neural networks, we use batches to train the model. This batches will be randomly shuffled, meaning that they do not necessarily succeed each other as they do in the time series. This step is undertaken because faster convergence has been observed if the order in which the mini-batches are visited is changed for each epoch, which can be reasonably efficient if the training set holds in computer memory [2].

3.3 COMPARISON

Our final goal will be to predict several one-day-ahead forecasts for each bond and model and compare them. The reason we pick to predict one day is due to the fact that predicting more days ahead while attaining usable result is unequally harder. As discussed below, the number of forecasts will be 20% of the according data sample. With the calculated forecasts at hand, we will calculate their error and check, whether the forecast has the same sign as the actual series, or not. This leads us to our first error measure which is called the **hit rate**:

$$\frac{1}{m}\sum_{i=1}^{m}\frac{|sign(x_i)+sign(\hat{x}_i)|}{2}$$

where x_i are the observed values, $\hat{x}_i$ our forecasts and m the number of forecasts generated for the concerning bond. When we later calculate the

hit rate for a number of bonds, we do this by taking the mean of the hit rates calculated bond-wise with the above formula.
Predicting the direction of the yield movement correctly, would be of great importance in real world applications.
Apart from the direction, we want to know how much our forecast differs from the actual value. Therefore we will also look at the root mean squared error **(RMSE)**

$$\sum_{i=1}^{m} \sqrt{\frac{(x_i - \hat{x}_i)^2}{m}},$$

where x_i are the correct values, $\hat{x}_i$ are the predicted values and m is the number of one-day forecasts made.

The third tool that we will use in order to compare our models will be the **Diebold-Mariano Test** [5]. It tests whether two forecasts have the same predictive power or are significantly different from each other.
Let $e_i = x_i - y_i^a$ and $r_i = x_i - y_i^b$ be the residuals per time step for the forecasts produced by method A: $Y^a = (y_1^a, y_2^a, \ldots, y_m^a)$ and the forecasts produced by metod B: $Y^b = (y_1^b, y_2^b, \ldots, y_m^b)$, where $X = (x_1, x_2, \ldots, x_m)$ are the observed values of that series. Further let $\bar{d}_i = \frac{1}{m}\sum_{i=1}^{m} d_i$ and set $d_i = e_i^2 - r_i^2$.
The **null hypothesis** of equal predictive accuracy of the forecasts, is:

$$\mathbb{E}(d_i) = \mu = 0.$$

To test for this hypothesis the Diebold-Mariano test (DM) computes

$$\gamma_k = \frac{1}{m}\sum_{i=k+1}^{m} (d_i - \bar{d})(d_{i-k} - \bar{d}) = \hat{Cov}(d_i, d_{i-k})$$

to finally get the test statistic:

$$DM = \frac{\bar{d}}{\sqrt{(\gamma_0 + 2\sum_{k=1}^{h-1}\gamma_k)/m}}$$

with h going to infinity. It is common practice to choose $h = m^{\frac{1}{3}} + 1$ however.
In the paper it is shown that under the null hypothesis, asymptotically we have: $DM \sim N(0,1)$. Therefore we reject the null hypothesis of equal predictability at the 5% level if $|DM| > 1.96$.
In Chapter 5 will apply this test to pairs of forecasts stemming from different models.
We will also test for the alternative null hypothesis: "method B is less accurate than method A" and "method B is more accurate than method A", corresponding to $\mathbb{E}(d_i) = \mu < 0$ and $\mathbb{E}(d_i) = \mu > 0$ respectively. Thus the null hypothesis gets rejected when $DM > 1.64$ in the first, and $DM < 1.64$ in the second case.

Now that we have introduced the various types of data that is available, we want to get more precise. When speaking of data we mean daily data, for 5 trading days per week. Out of computational-power restrictions we limit this thesis to daily, rather than hourly data. As outlined in [10] intra day stock market data can be very valuable, especially when news about a possible takeover, acquisition or merger [MRG] are circulated. In a 10 minute window, starting with the minute of the news alert about an MRG event, significant stock price drifts can be observed. Although bonds are traded much less frequently than stocks, this may also be true for bonds, but is not further examined here.
The data set consists of 91 bonds. Those are the bonds remaining after selecting only bonds with at least 800 days of data of all in Chapter 1 mentioned types. Every dataset is then split into a training-, a validation- and a test-set according to the ratio: (40%/40%/20%).
The following procedure will be applied to all bonds to try to predict the bond's spread changes:

1. **Choosing Hyperparameters**
    - Train the model on the first 40% of the data
    - Make one day forecasts for the datapoints of the next 40% of the data. Compare those forecasts with the given spread changes and adjust the model's hyperparameters that achieve the highest hit rate. In case the hit rates of two hyperparameter choices are similar, choose the hyperparameters achieving the lowest RMSE.
2. **Forecasting**
    - Train the model with the chosen hyperparameters on the first 80% of the data
    - Make one day ahead forecasts, predicting the datapoints in the remaining 20% of the data.

For choosing the hyperparameters, we train and test the model on the first 80% of the data and leave the last 20% untouched in order to prevent overfitting. Those last 20% are also not shown to the model when training it for the final forecast, and are only used to calculate the model error E.

4 THE MODELS

4.1 ARMA-GARCH

Due to the recurring short periods of high volatility, it is natural to try modelling the time series via an ARMA process with GARCH errors(ARMA-GARCH). But to verify the applicability of this model, we first fit an ARIMA sequence, and use the Ljung-Box test to check the residuals and the squared residuals of this ARIMA model for autocorrelations.
For non-squared residuals the hypothesis of independently distributed residuals gets rejected for all but 3 datasets (23, 68, 69), having p-values bigger than 0.05 $(0.39, 0.40, 0.60)$. For the squared residuals only dataset 76 seems to have no noteworthy correlations with a p-value of approximately 0.12.

Given the the availability of additional explanatory variables, we will use a modified ARMA-GARCH model that allows for external regressors.

Definition 4.1. *A process $(X_t)_{t\in\mathbb{Z}}$ follows an **ARMA-GARCH** model with an external regressor v_t, if it is an ARMA process with errors ϵ_t that satisfy:*

$$\epsilon_t = z_t\sigma_t,$$

where $(z_t)_{t\in\mathbb{Z}}$ is an iid process with zero mean and unit variance and σ_t is the conditional variance that satisfies the modified GARCH equation:

$$\sigma_t^2 = c + \zeta v_t + \sum_{j=1}^{q} \alpha_j \epsilon_{t-j}^2 + \sum_{j=1}^{p} \beta_j \sigma_{t-j},$$

where $c, \zeta, \alpha_1, \ldots, \alpha_q$ and $\beta_1, \ldots, \beta_p$ are constants. As is the case with the variance term, we also admit an external regressor in the ARMAX process as in definition 2.6.

THE IMPLEMENTATION of the model was done in R using the rugarch package [1].
As the variables containing information about the news events, showed by far the strongest correlation to the spread change, we will try to exploit this relationship. Therefore we selected the variable marking 2 days prior and two days after a news event with an entry 1 as our first exogenous variable. We will then also use the sentiment variable, to subsequently compare the results of using either of them.
For this model the following hyperparameters need to be chosen:

- ARMA orders **P** and **Q**

1 To be found online at https://cran.r-project.org/web/packages/rugarch/rugarch.pdf/.

- GARCH orders **p** and **q**

We now want to loop through a set of predetermined parameter combinations, fit our ARMA-GARCH model on the validation set, perform forecasts and find the combination of hyperparameters whose forecasts give us the best hit rate. This procedure will be repeated for every of the 91 datasets, for every of our two chosen explanatory variables and every of our two approaches on using external data. We determine the set of possible parameters as: $P \in \{4,5,6\}$, $Q \in \{4,5,6\}$, $p \in \{2,3\}$ and $q \in \{2,3\}$.

We start by using the **exogenous data in the variance**.
Using news as exogenous data, the achieved results are very poor with the best performing parameter combination P:4, Q:5, p:3, q:3, achieving a hit rate of 0.498738, meaning the model's result is not different from a random draw. When choosing the sentiment instead of the news data, the best hit

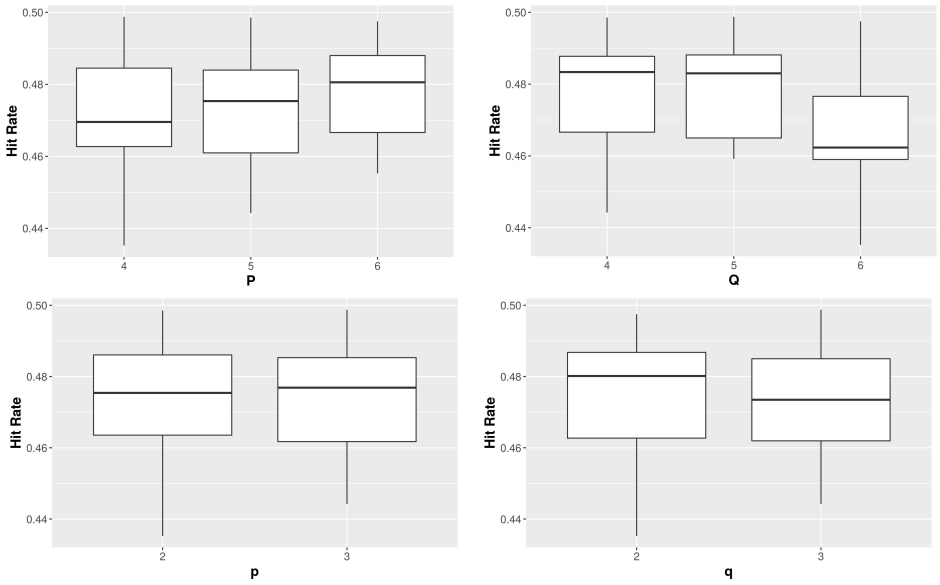

Figure 4.1: Plots showing the hit rates achieved by hyperparameter combinations containing the respective parameters, when using news as exogenous data in the variance

rate 0.5104407 gets achieved by P:4, Q:5, p:2, q:3. Figure 4.1 and Figure 4.2 show, that choosing higher ARMA and GARCH orders, does not necessarily lead to better results. Also there seems to be little correlation between the selected orders and the hit rate, apart from the order $P = 4$, which is involved in the best hitrate and stands out in our boxplot for the sentiment.

After looking at hyperparameters and their performance for all datasets simultaneously, we now want to look at groups of bonds issued by the same company. We hope to get an improved forecasting performance from choosing the same set of hyperparameters for those groups, without running the risk of overfitting, which we would do by determining different hyperparameters for every bond in our sample. In Chapter 1 we saw that bonds of the same company exhibit similar behaviour, which supports this approach.

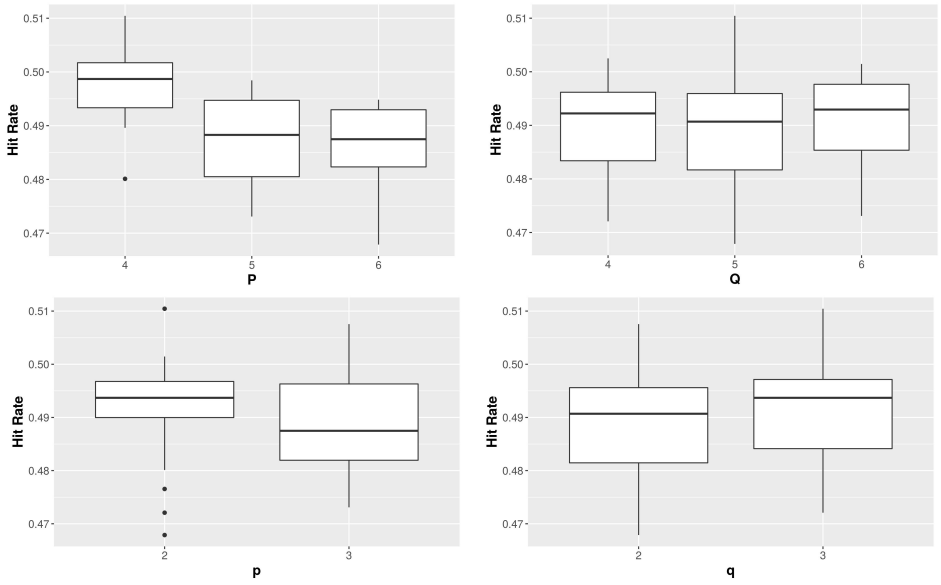

Figure 4.2: Plots showing the hit rates achieved by hyperparameter combinations containing the respective parameters, when using sentiment as exogenous data in the variance

P	Q	P	Q	**hit rate**	**company**
5	5	2	3	0.55072	Intesa Sanpaolo SPA
6	4	3	3	0.53544	Societe Generale Group
4	5	2	3	0.52631	BNP Paribas
5	5	3	2	0.52238	Adidas
4	5	3	2	0.50815	Deutsche Bank AG

Table 4.2: Top Hyperparameter Choices when using Sentiment

Table 4.2 and Table 4.4 show the best hit rates for every company and the used hyperparameters. There are notable differences in the predictability of the companies, with "Deutsche Bank AG" the hardest to predict. We also note, that the fact that these hit rates are much higher, than those achieved above without grouping, is due to the smaller set of bonds that receive individual hyperparameters. Looking at the exogenous variables, there seems to be no clear better choice, with the sentiment performing slightly better.

We now want to consider the best hyperparameter choices for every dataset separately. Table 4.6 shows that the hit rates increase dramatically and that the datasets with the best hit rates do not coincide for the two exogenous variables. This is true not only for 3, but the top 7 data sets. In Chapter 5 we will see whether or not, determining the hyperparameters for every dataset individually, will lead to overfitting.

P	Q	P	Q	hit rate	company
6	4	3	3	0.54888	Societe Generale Group
4	5	2	3	0.53899	Intesa Sanpaolo SPA
6	4	2	3	0.53409	Adidas
5	4	2	2	0.52445	BNP Paribas
4	4	3	2	0.51094	Deutsche Bank AG

Table 4.4: Top Hyperparameter Choices when using News

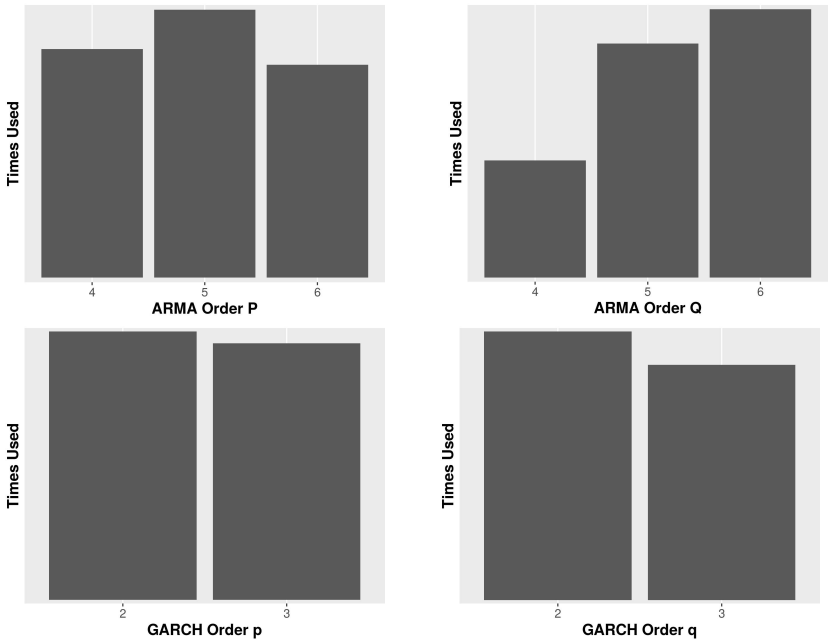

Figure 4.3: Number of Times the different Hyperparameters were chosen when using News as the Explanatory Variable

As expected, picking a higher ARMA-GARCH order, does not necessarily improve the average hit rate, when using either of the explanatory variables. For the case of utilizing the news variable Figure 4.3 illustrates this behaviour.

VARIABLE	dataset nr.	P	Q	P	Q	hit rate
Sentiment	4	4	6	3	3	0.64012
	76	4	4	2	3	0.62864
	81	5	5	2	2	0.61388
News	70	6	4	3	3	0.64634
	74	5	5	2	2	0.61604
	55	5	5	2	3	0.61338

Table 4.6: Top 3 Hit Rate Results

What follows is the same proceeding as above, this time using the **exogenous variable in the mean**.
Using news as exogenous data, the achieved results again are very poor with the best performing parameter combination P:6, Q:4, p:3, q:2, achieving a hit rate of 0.462893, which is not better than a random draw.

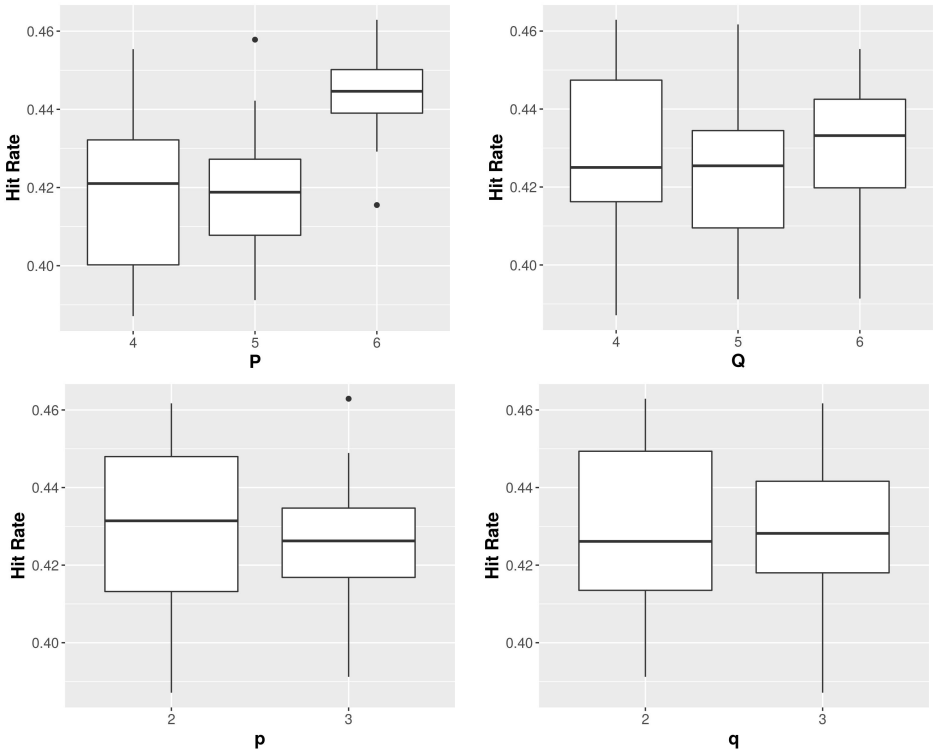

Figure 4.4: Plots showing the hit rates achieved by hyperparameter combinations containing the respective parameters, when using news as exogenous data in the mean

When choosing the sentiment instead of the news data, the best hit rate 0.4618602 gets achieved by P:6, Q:5, p:3, q:3.
Those results are both significantly lower as was the case when using the exogenous variables in the variance.
Figure 4.4 and Figure 4.5 show, that in the case of an external regressor in the mean, choosing higher ARMA orders, seems to lead to slightly better results, which cannot be observed for the GARCH orders.

Again we now want to look at sets of hyperparameters for the bonds grouped by the emitting companies. In Table 4.8 and Table 4.10 we see the results for the companies. It is again evident that with this model bonds of "Deutsche Bank AG" are the hardest to predict, while for the others, no clear pattern is observable. The exogenous variable news seems to give us slightly less explanatory power than we got when using the ARMA-GARCH model with the external regressor in the variance term, the contrary is true for the exogenous variable sentiment.

Again we want to consider the best parameter choices for every single dataset. Table 4.12 shows that the hit rates increase dramatically and that

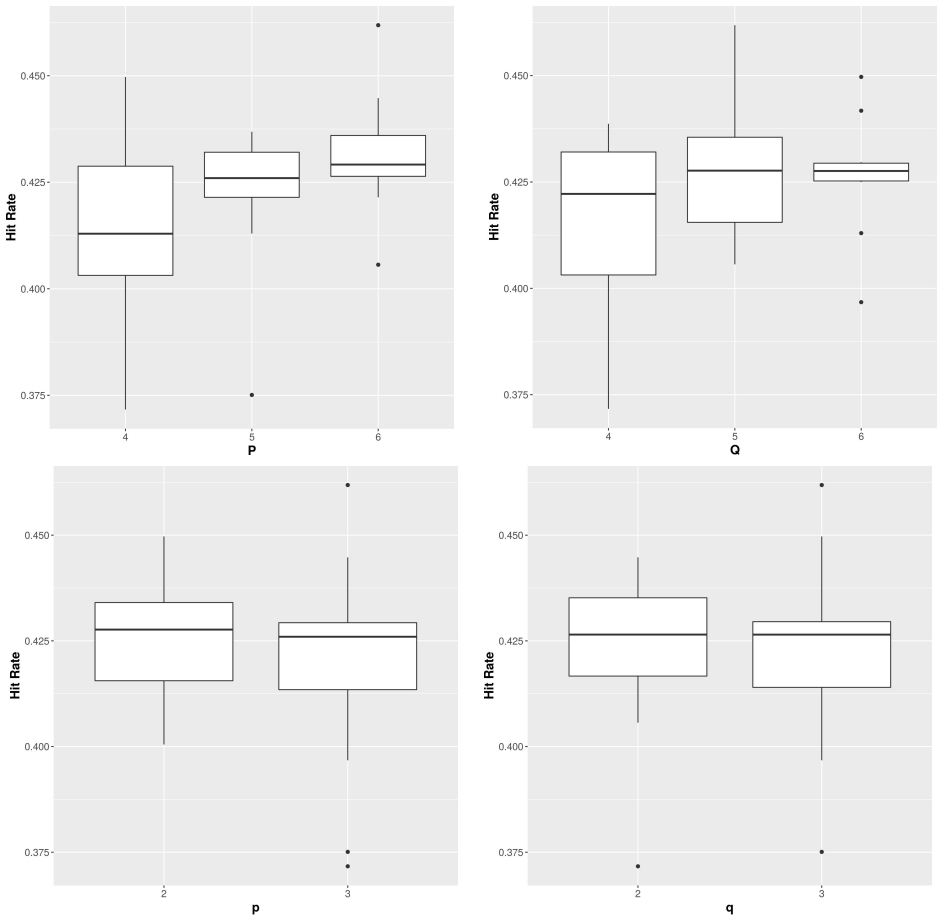

Figure 4.5: Plots showing the hit rates achieved by hyperparameter combinations containing the respective parameters, when using sentiment as exogenous data in the mean

the datasets with the best hitrates do partly coincide for the two exogenous variables, in contrast to what we observed for the exogenous variable in the variance. Out of the first 6 datasets, 4 coincide.
In Chapter 5 we will see whether or not, determining the hyperparameters for every dataset individually, will lead to overfitting.

As expected, picking a higher ARMA-GARCH order, does not necessarily improve the average hit rate, when using either of the explanatory variables. For the case of utilizing the news variable Figure 4.6 illustrates this behaviour.

P	Q	P	Q	hit rate	company
4	6	2	3	0.57801	Adidas
4	4	3	2	0.52923	Societe Generale Group
6	5	2	3	0.52520	Intesa Sanpaolo SPA
6	6	3	2	0.52506	BNP Paribas
6	4	3	3	0.51182	Deutsche Bank AG

Table 4.8: Top Hyperparameter Choices when using Sentiment

P	Q	P	Q	hit rate	company
4	5	2	3	0.53817	Intesa Sanpaolo SPA
6	4	3	3	0.53594	BNP Paribas
4	6	2	2	0.53192	Adidas
6	6	3	2	0.52821	Societe Generale Group
6	5	2	2	0.50870	Deutsche Bank AG

Table 4.10: Top Hyperparameter Choices when using News

4.2 STAR

Another model that makes use of an exogenous variable, is the smooth transition autoregressive(STAR) model. It is an extension of the AR model 2.4, that allows the parameters of the AR model to change depending on an exogenous "transition" variable. It can be thought of as a set of arbitrary many AR models, of which one will be chosen, depending on the threshold or transition variable Z_t and the parameters γ and th.

VARIABLE	dataset nr.	P	Q	P	Q	hit rate
Sentiment	74	5	4	2	3	0.61071
	65	4	4	2	2	0.60744
	88	6	5	2	3	0.60417
News	74	5	4	2	3	0.63571
	21	6	6	2	3	0.62376
	88	6	4	3	2	0.609375

Table 4.12: Top 3 Hit Rate Results

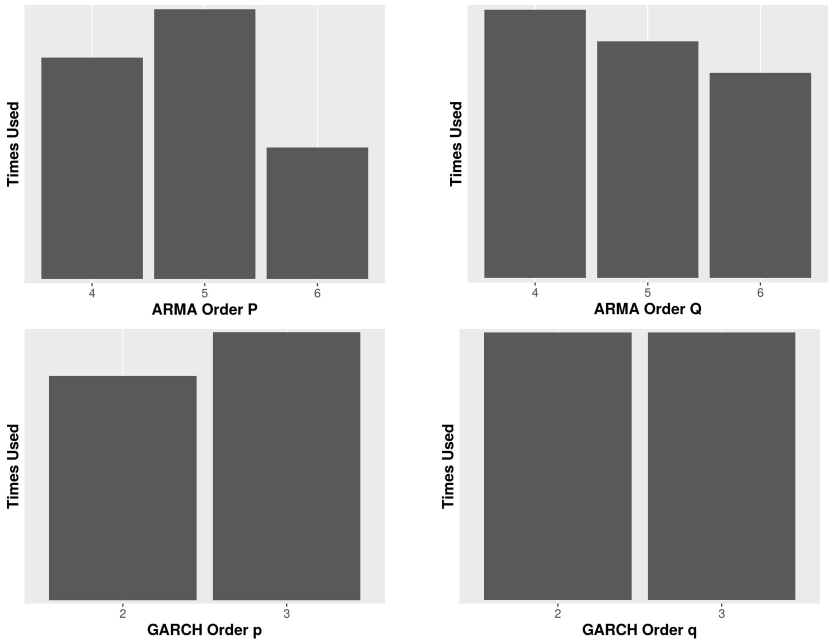

Figure 4.6: Number of Times the different Hyperparameters were chosen when using News as the Explanatory Variable

Definition 4.2. *A process* $(X_t)_{t\in\mathbb{Z}}$ *follows a* ***STAR*** *model if:*

$$X_{t+s} = (\phi_1 + \phi_{10}X_t + \phi_{11}X_{t-1} + \ldots + \phi_{1p}X_{t-p})(1 - G(Z_t, \gamma, th)) + \tag{4.1}$$

$$(\phi_2 + \phi_{20}X_t + \phi_{21}X_{t-1} + \ldots + \phi_{2p}X_{t-p})G(Z_t, \gamma, th) + \epsilon_{t+s} \tag{4.2}$$

holds for the parameters $p, \phi_1, \phi_2, \phi_{1i}, \phi_{2j}$, $\forall 0 \leq i \leq mH \quad \forall 0 \leq j \leq mL$ *aswell as for the transition function* $G(Z_t, \gamma, th)$ *that is bounded between* 0 *and* 1*, a white noise sequence* ϵ_t*, the exogenous transition variable* Z_t *and the real valued parameters* th *and* $\gamma > 0$*.*

For the function $G(Z_t, \gamma, th)$ several choices are possible, the most popular are first and second order logistic functions. We will use the first order logistic function. Note that using Definition 2.13 we have $G(Z_t, \gamma, th) = f(Z_t)$ where γ corresponds to k, th corresponds to x_0 and we set $L = 1$ in order to achieve a maximum value of 1.

THE IMPLEMENTATION of STAR models was done in R using the tsDyn package [2]. The procedure to find a suitable model is, to first choose an exogenous sequence, and then to estimate the parameters. In our example, we perform the estimation of the model by using the BFGS algorithm [4], with the analytical gradient.
As with the ARMA-GARCH model we will try to use sentiment and news as exogenous variables. We select the news variable counting the days

2 To be found online at https://cran.r-project.org/web/packages/tsDyn/index.html.

passed since the last news event as our first, and the sentiment as our second variable, and subsequently compare the results of using either of them. For this model the following hyperparameters need to be chosen:

- **mH**, the autoregressive order of the "high" regime
- **mL**, the autoregressive order of the "low" regime

We now want to loop through a set of predetermined hyperparameter combinations, fit our STAR model on the validation set, perform forecasts and find the combination of hyperparameters whose forecasts give us the best hit rate. For every of our 91 datasets, and both our explanatory variables, this procedure will be repeated. We determine the set of possible parameters as: mH $\in \{4,5,6\}$ and mL $\in \{4,5,6\}$.
Using news as exogenous data, the achieved results were good, with the best performing parameter combination mH:4, mL:4, achieving a hit rate of 0.5292783.

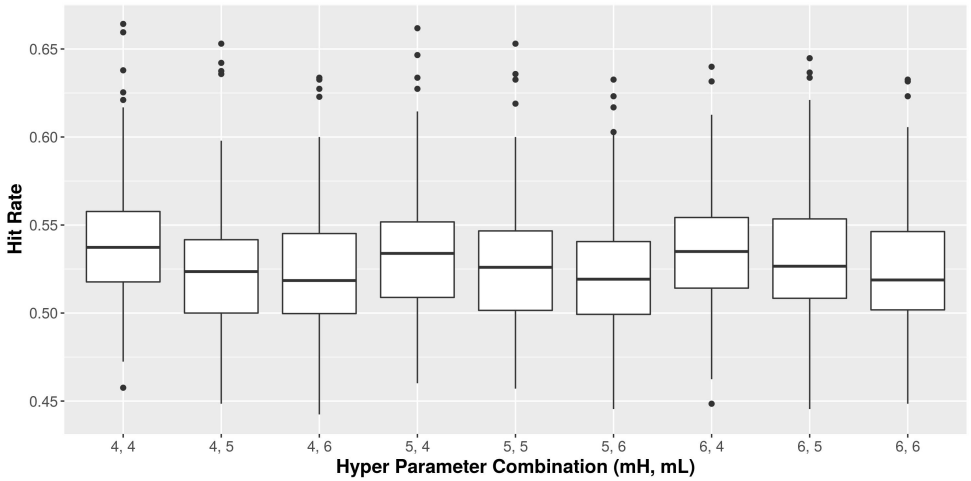

Figure 4.7: Boxplot showing the hitrates achieved with news as an explanatory variable for the 9 chosen parameter combinations.

When choosing the sentiment instead of the news data, the best hit rate 0.5419361 gets achieved by mH:5, mL:6. As with the ARMA-GARCH model, it is clearly above the hit rate of the news data. Figure 4.7 shows, that using news, the preferred mL-order is 4, whereas there is no clear pattern visible for the results for different mH-orders. This means that lower autoregressive orders can result in better hit rates, rather than the intuitively chosen higher ones. The situation is the opposite with sentiment, where higher mL-orders tend to yield better hit rates. Here also higher mH-orders seem to improve the outcome.

The next step of the validation process will be to take a look at hyperparameters for groups of bonds issued by the same company. Again we hope to get an improved forecasting performance from choosing the same set of hyperparameters for those groups.

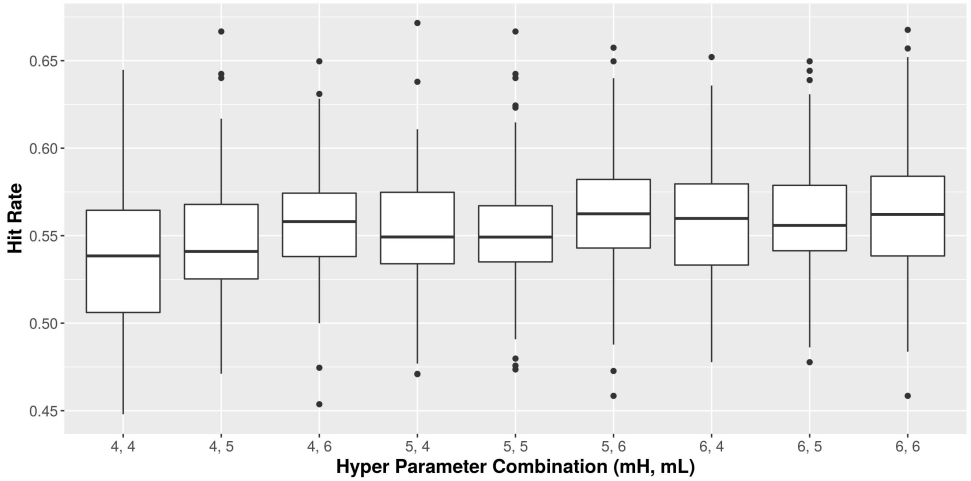

Figure 4.8: Boxplot showing the hitrates achieved with sentiment as an explanatory variable for the 9 chosen parameter combinations.

MH	ML	hit rate	company
4	4	0.55996	Intesa Sanpaolo SPA
6	5	0.63888	Societe Generale Group
6	6	0.57378	BNP Paribas
5	6	0.54896	Adidas
6	5	0.54976	Deutsche Bank AG

Table 4.14: Top Hyperparameter Choices when using Sentiment

Table 4.14 and Table 4.16 show the best hit rates for every company and the used hyperparameters. As in the section about the ARMA-GARCH models, we see a notable higher predictability for "Societe Generale", with "Adidas" and "BNP Paribas" being far behind. "Deutsche Bank" again seems to be hard to predict, but as the average hit rate is higher now, the results are still very strong.
This approach again yields higher hit rates as when taking one set of hyperparameters for all bonds, which is due to the smaller set of hit rates we calculate the average of when choosing the parameters.

MH	ML	hit rate	company
6	6	0.60555	Societe Generale Group
4	4	0.56677	BNP Paribas
4	5	0.55621	Adidas
4	4	0.54216	Deutsche Bank AG
4	4	0.51009	Intesa Sanpaolo SPA

Table 4.16: Top Hyperparameter Choices when using News

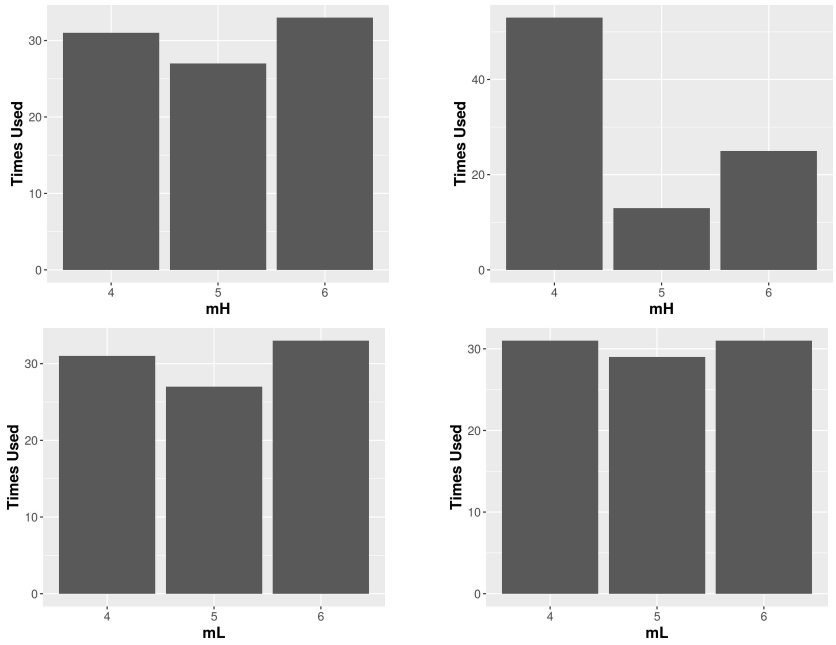

Figure 4.9: Number of times the different hyperparameters were chosen when using news (right) and sentiment (left) as the explanatory variable

The last step of validation will be to find the best parameter choices for every single dataset, without any grouping. Table 4.18 shows that the hit rates increased a lot compared to above. In contrast to the ARMA-GARCH model, a lot of well performing datasets coincide for the two exogenous variables.
Again, a higher ARMA-GARCH order, does not necessarily improve the average hit rate, when using either of the explanatory variables. In our example, there are no clearly preferred parameters. See Figure 4.9 to observe this behaviour.

VARIABLE	dataset nr.	MH	ML	hit rate
Sentiment	53	6	6	0.66760
	51	6	6	0.65697
	76	6	6	0.65206
News	74	4	6	0.66424
	75	4	5	0.66423
	76	4	4	0.66398

Table 4.18: Top 3 Hit Rate Results

4.3 LSTM

In this section we will introduce the 'long short-term memory network' (LSTM). It belongs to the class of artificial neural networks (ANN).

VANISHING / EXPLODING GRADIENTS: In [6] Hochreiter showed that for "Backpropagation Through Time"(BPTT) and "Real-Time Recurrent Learning"(RTRL) by Robinson and Fallside[1] the weights of the ANN are not all equally valued, meaning the updates the weights w_{ij} receive, are proportional on the gradient of the error function with respect to the current weight in each iteration of training. This results in problems when choosing the weights via backpropagation, as w_{ij} can have too big or too little impact on the gradient of the error function. This leads to 'blowing up'(exponential growth) or 'vanishing' (exponential decrease) of the gradients. With too large gradiendts, we will hardly find a value for weights that yield a satisfactory error, whereas with too small gradients, the search for a satisfactory result will be considerably slower.

SOLUTION: This problem gets eliminated with the concept of long-short-term networks, first introduced in [12]. It is a novel recurrent network architecture with a gradient based learning algorithm. LSTM can bridge time intervals of over 1000 steps, in case of noisy sequences, without loosing the ability to capture short term events.
It consists of units, called memory cells. Those units build an artificial neural network and are called LSTM-Networks. The difference to conventional artificial neural networks lies in the nature of the LSTM-units compared to nodes of standard neural network. A node as described in Chapter 2 does contain an activation and an activation function. A memory cell j, of a LSTM network with m inputs and n cells, however consists of the following parts:

1. **input gate:** It protects the cells stored information. Its activation follows the equality:
$$a_j^{in}(t) = \sigma(\underbrace{\sum_{k=1}^{m} w_{jk}^{in} x_k(t) + \sum_{u=1}^{n} v_{ju}^{in} a_u(t-1) + b_j^{in}}_{net_j^{in}(t)})$$
2. **output gate:** It protects other cells from irrelevant input. Its activation follows:
$$a_j^{out}(t) = \sigma(\underbrace{\sum_{k=1}^{m} w_{jk}^{out} x_k(t) + \sum_{u=1}^{n} v_{ju}^{out} a_u(t-1) + b_j^{out}}_{net_j^{out}(t)})$$
3. **forget gate:** controls how much of the previous cell state will be forgotten: $a_j^f(t) = \sigma(\underbrace{\sum_{k=1}^{m} w_{jk}^{f} x_k(t) + \sum_{u=1}^{n} v_{ju}^{f} a_u(t-1) + b_j^{f}}_{net_j^{f}(t)})$

4. **internal cell state:** which follows:

$$a_j^{state}(t) = a_j^f \; a_j^{state}(t-1) + a_j^{in}(t) \; tanh(\underbrace{\sum_{k=1}^{m} w_{jk}^{state} x_k(t) + \sum_{u=1}^{n} v_{ju}^{state} a_u(t-1) + b_j^{state}}_{net_j^{state}(t)})$$

5. **cell output:** that is computed with the help of the gates via:
   $a_j(t) = a_j^{out}(t) \; tanh(a_j^{state}(t))$,

where σ is the sigmoid function 2.14, $x(t) \in \mathbb{R}^m$ for $1 \leq t \leq T$ is some exogenous input, and $v_{ju}^i, w_{jk}^i, b_j^i \in \mathbb{R}$ with $i \in \{$"out", "in", "f", "state"$\}$, are the parameters to learn.
We can see that the gates are "conventional" neurons as seen in the feedforward neural network, but get some past values of the cell's output. The cell state even uses its own time delayed outputs as inputs. This interpretation leads us to Figure 4.10, depicting a single LSTM-cell.

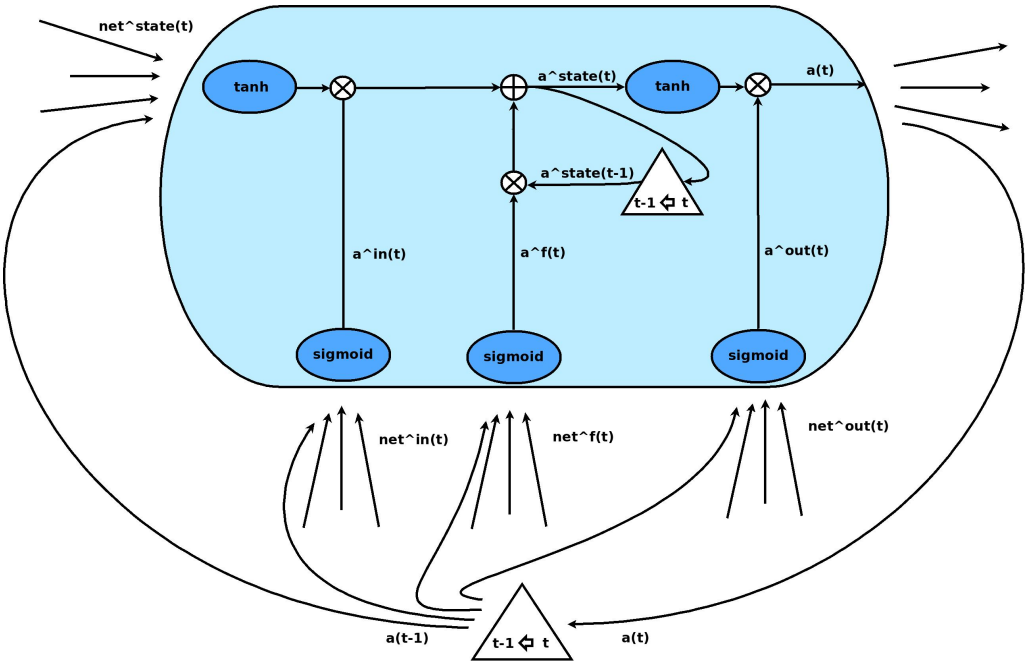

Figure 4.10: Architecture of a single LSTM-cell

The equations for the entire LSTM network with n hidden units, and an input vector $x(t) \in \mathbb{R}^m$ for $1 \leq t \leq T$, can be described with the following equations.

1. $a^{in}(t) = \sigma(\underbrace{W^{in} x(t) + V^{in} a(t-1) + b^{in}}_{net^{in}(t)}) \quad \in \mathbb{R}^n$

2. $a^{out}(t) = \sigma(\underbrace{W^{out}x(t) + V^{out}a(t-1) + b^{out}}_{net^{out}(t)}) \in \mathbb{R}^n$

3. $a^{f}(t) = \sigma(\underbrace{W^{f}x(t) + V^{f}a(t-1) + b^{f}}_{net^{f}(t)}) \in \mathbb{R}^n$

4. $a^{state}(t) =$
   $a^{f}(t)\ a^{state}(t-1) + a^{in}(t)\ tanh(\underbrace{W^{state}x(t) + V^{state}a(t-1) + b^{state}}_{net^{state}(t)}) \in \mathbb{R}^n$

5. $a(t) = a^{out}(t)\ tanh(a^{state}(t)) \in \mathbb{R}^n$,

where σ is the sigmoid function 2.14, $x(t) \in \mathbb{R}^m$, $W^i \in \mathbb{R}^{n\times m}$, $V^i \in \mathbb{R}^{n\times n}$ and $b^i \in \mathbb{R}^n$. Note that the gates of a LSTM-unit are conventional artificial neurons. For another graphical representation of a single LSTM-cell see Figure 4.11.

Strictly speaking the above network is a LSTM-network with a forget gate, as this concept was added later. It has been observed that long continuous input streams, without explicitly marked ends at which the network's internal states can be reset, can make the states to grow indefinitely. To circumvent this problem, in [12] this variant of the LSTM network was presented, introducing the forget gate. It controls how much information from the previous cell state should be forgotten. Also note that the forget gate is computed just like the input and the output gate and therefore is a neuron of an ordinary feedforward neural network.
Another variant of the LSTM network would be to replace $a(t-1)$ with $a^{state}(t-1)$ for the first three equations above. This allows the gates to have a look at the cell state, motivating the name Peephole LSTM. We will however no longer pursue this approach.

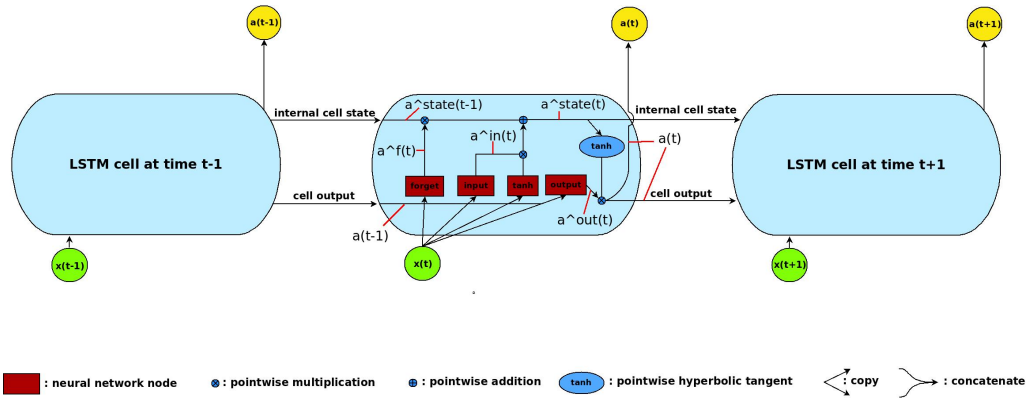

Figure 4.11: Structure of a single LSTM-cell

THE IMPLEMENTATION of an LSTM network has been performed in the Pytorch environment of Python[3].
In order to produce a sequence of outputs $(d(1), \ldots, d(P-1))$, the LSTM network processes a sequence of inputs $(x(1), \ldots, x(P-1))$. The architecture of the LSTM network which we use is displayed in Figure 4.12 . We use one or two hidden layers where each of these layers consists of n LSTM units. n is called the **hidden size**.
Our goal is to compute a prediction $\hat{y}(t)$ for the spread change $y(t)$. To this end we use our explanatory variables $x(t) \in \mathbb{R}^m$, where $x(t)$ also contains the lagged spread change $y(t-1)$. To compute the prediction we present the sequence of the last $(P-1)$ values of x, i.e. $(x(t-P+2), \ldots, x(t))$ to the LSTM network and get a sequence of n-dimensional outputs $(d(t-P+2), \ldots, d(t))$. The prediction finally is a linear combination of these outputs:

$$\hat{y}(t) = M^\top d + b,$$

where $d = (d(t-P+2)^\top, \ldots, d(t)^\top)^\top \in \mathbb{R}^{(P-1)n}$, $M \in \mathbb{R}^{(P-1)n}$, $b \in \mathbb{R}$.

We use minibatch stochastic gradient descent (SGD), together with the Adam optimizer [8] 2.1 for the training. As the objective function we chose the mean squared error:

$$\frac{1}{|N|} \sum_{t \in N} (\hat{y}_t - y_t)^2$$

with N being the indices of our random batch, $|N|$ being the batch size, $\hat{y}_t$ being the predicted and y_t the observed value.

3 To be found online at http://pytorch.org/.

We now want to tackle the selection of the hyperparemeters for our model, which are:

1. learning rate α
2. decay rates for the Adam optimizer β_1, β_2
3. number of epochs
4. batch size
5. numer of hidden layers
6. hidden size n
7. time horizon P
8. set of exogenous variables

For the selection of hyperparameters $1-7$ all available exogenous variables will be used as input. After choosing the hyperparameters $1-7$ we will determine the set of exogenous variables that delivers the best predictive power. To do that, we will use the predetermined hyperparameters $1-7$ and different combinations of exogenous variables in order to make predictions with our LSTM model. For every of the possible combinations of exogenous variables we will create a set of predictions. The set of explanatory variables that was used to make the set of predictions having the best average hit rate will get selected, and will later be used for our final predictions.

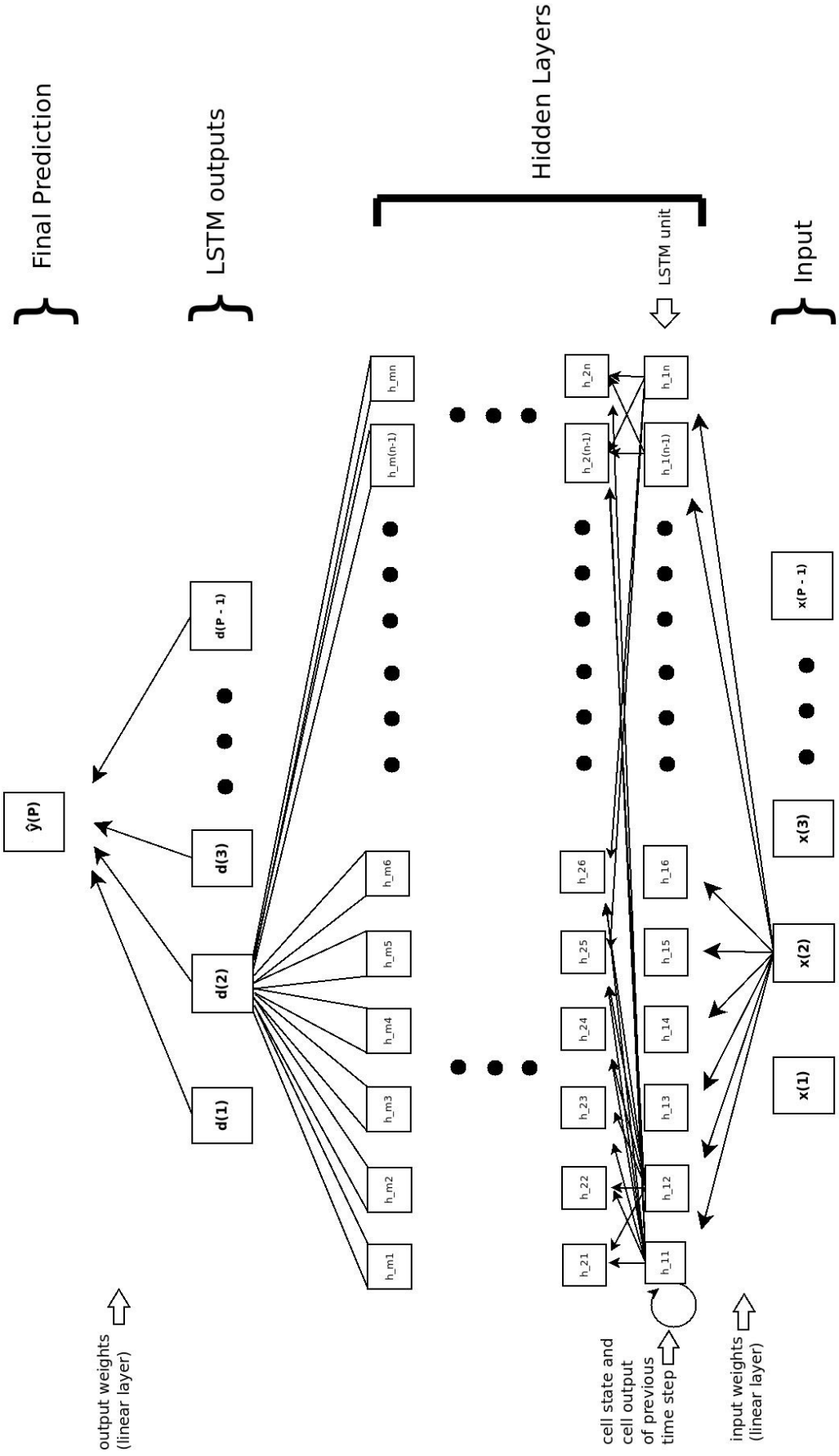

Figure 4.12: LSTM network architecture

Very good results could be achieved with a learningrate starting with 0.01. For the decay rate we did adopt the widely used choice: $\beta_1 = 0.9, \beta_2 = 0.999$, which we will also use for the DA-RNN network below.
With these hyperparameters fixed, the remaining ones were determined by validation. This process consisted of first predetermining a certain range of possible hyperparameters to choose from, training the model with those hyperparameters, generating forecasts and subsequently comparing the hit rates, achieved by the different hyperparameter settings. The possible hyperparameter ranges were picked as: number of hidden layers $\in \{1, 2\}$, hidden size $\in \{16, 32, 64, 128\}$, batch size $\in \{16, 32, 64\}$, time horizon $\in \{16, 32, 64\}$. For validating the network, we made use of all avaliable variables
The best result, was achieved with 2 hidden layers. Using: batch size 64, hidden size 16 and a time horizon of 8 the computation resulted in a hit rate of 0.5280215 with a RMSE of 0.1314439. When pursuing trading strategies making use of those results, the benefits of a slightly higher hit rate may outweigh the costs of extra computational power by far. As we have only restricted computational resources at hand, the increase of the hit rate does not justify the extra time needed to train the model in our case. That is why we will not pursue testing models with more than 2 hidden layers.

As the two layer model outperformed the one layer model, we will use the former to conduct our final forecasts. We still want to see the differences between those two models and did validate them separately. We will start our analysis with the one layer model. The top three parameter choices were:

BATCH SIZE	HIDDEN SIZE	TIME HORIZON	RMSE	**hit rate**
32	16	64	2.4696204	0.5239922
64	64	16	3.5238382	0.5232146
16	128	16	1.9893613	0.5214029

Table 4.20: Top 3 Hyperparameter Choices

The performance for all other parameter settings not belonging to the top 3, was significantly worse. We will additionally consult boxplots for every parameter, showing the distribution of the hit rate for different parameter values, to make our final parameter choice.

Figure 4.13 shows, that results for our data get worse with increasing batch size and confirms our intuitive assumption that the model gets more accurate with a higher hidden size, as it is able to pick up more correlations. We however have to be cautious and should refrain from assuming this holds for arbitrary models and arbitrary hidden sizes. The lower average hit rate at hidden size 128 shows that high hidden sizes can lead to overfitting or very slow learning models that would need much more epochs to train in order to perform good results. The lower left part of Figure 4.13 de-

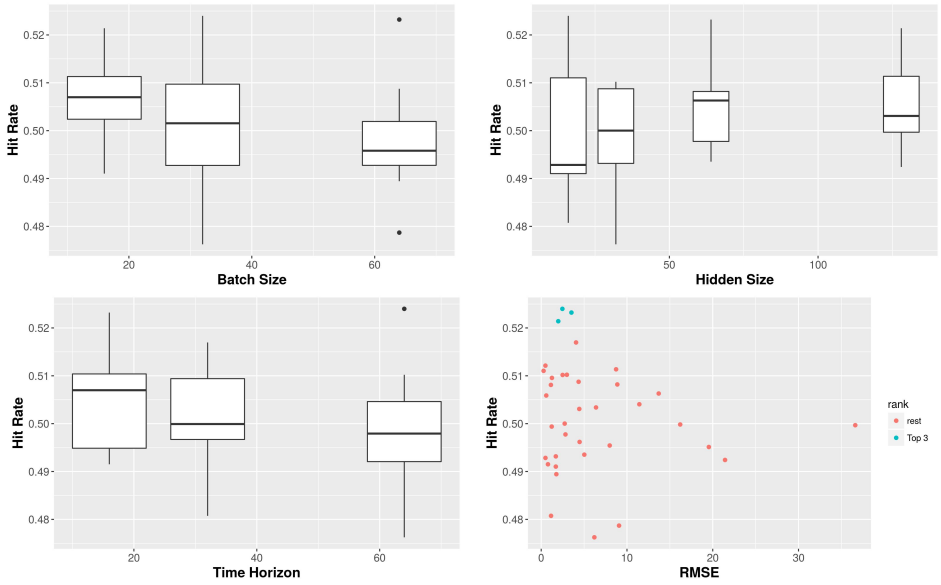

Figure 4.13: Plots for the different hyperparameters of the LSTM network with 1 hidden layer

livers more surprising insights. An increasing time horizon seems to have a negative influence on our hit rate. Again this behaviour might change with an higher number of epochs, and shouldn't be generalized. Finally the figure reveals the superiority of the best 3 results for this model, having a low error combined with a good predictive performance.
This motivates the following parameter choice: batch size = 16, hidden size = 128, time horizon = 16, as this is a top 3 result with only slight worse hit rate but better RMSE values compared to the other 2 results.
Increasing the number of layers to 2, the plots of Figure 4.14 look fairly similar to the results for the one layer model. The parameter setting achieving by far the best values (batch size 64, hidden size 16,time horizon 8), will be used for the final forecast.

Until now, we did look at the results when choosing a certain hyperparameter set for all datasets in our sample. As with the ARMA-GARCH model, we now want to group the bonds into batches of the same company. Table 4.22 reveals a higher predictability of Societe Generale with our LSTM model. Earlier we observed similar behaviour when applying the ARMA-GARCH model. The top 10 parameter choices for this company result in higher hit rates than any of the other companies achieved.

BATCH	HIDDEN SIZE	P	**hit rate**	**company**
32	128	32	0.70897	Societe Generale
16	16	16	0.55394	Intesa Sanpaolo
16	64	16	0.54514	Adidas
16	16	8	0.53064	Deutsche Bank
64	32	32	0.52925	BNP Paribas

Table 4.22: Top Hyperparameter Choices per Company

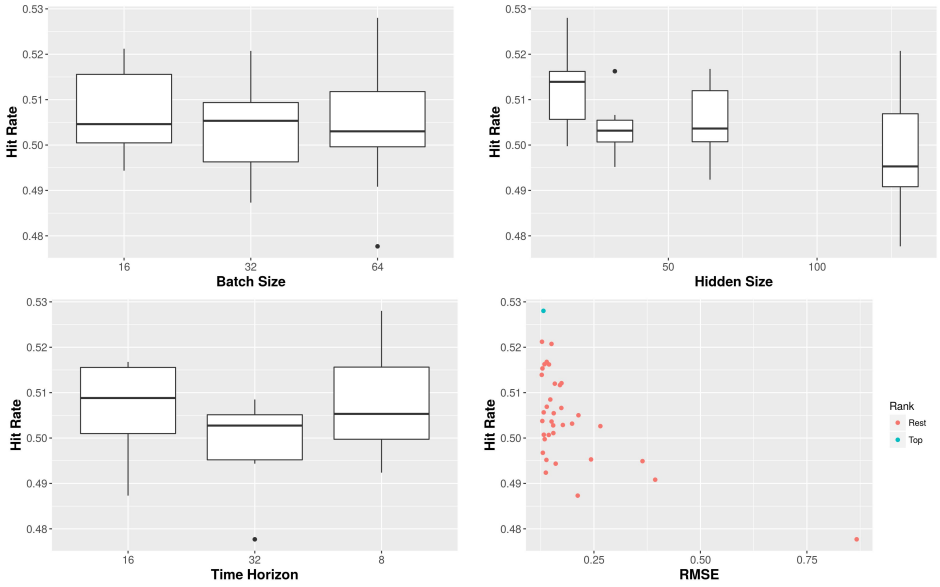

Figure 4.14: Plots for the different hyperparameters of the LSTM network with 2 hidden layers

As in the ARMA-GARCH case, we now proceed with looking at the best parameters for every dataset individually. Table 4.24 depicts the top 3 hit rates, notably resulting from data of 2 bonds of Societe Generale and 1 of Deutsche Bank. The lowest hit rate is 0.52559, which is still well above the 0.5 mark.

dataset nr.	BATCH SIZE	HIDDEN SIZE	P	hit rate
66	16	128	32	0.787582
65	64	32	8	0.758993
21	16	128	32	0.701205

Table 4.24: Top 3 Hit Rate Results

Figure 4.15 promotes a high number of hidden nodes and shows some ambiguity concerning the time horizon to choose. This shows again, that choosing a model that can potentially capture more information, is not always preferable. It is also evident that smaller batch sizes perform better on average.

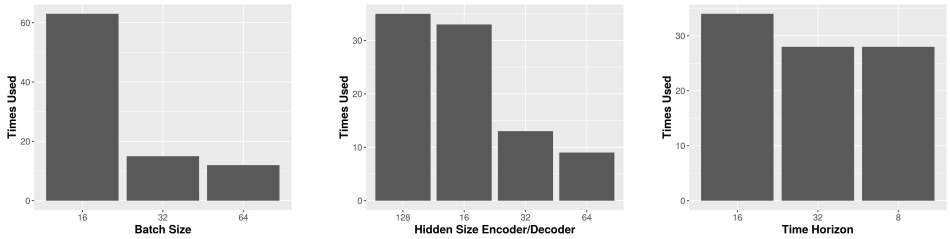

Figure 4.15: Number of Times the different Hyperparameters were chosen

With the parameters chosen, we want to take another look at the exogenous variables. The readily configured model, gives us a tool to measure the different variables' prediction power, complementing the methods we did apply in Chapter 1.

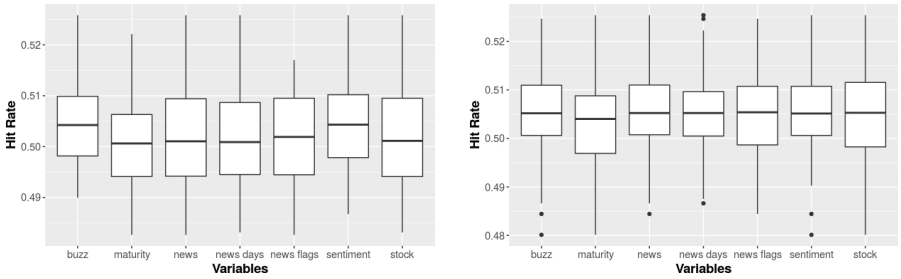

Figure 4.16: Boxplot of different hyperparameters and the achieved hit rates of forecasts that make use of those hyperparameters, using 1 (left) resp. 2 layers (right)

Figure 4.16 plots the hit rates resulting of the set of hyperparameters determined above, and a set of explanatory variables. All combinations of explanatory variables were trained, and the resulting hit rates are plotted for all variables that were used for the respective forecast. It shows that only the time to maturity differs significantly from the other exogenous variables, as it seems to bring no advantage to our forecast model at an average hit rate of approximately 0.5 for forecasts resulting from hyperparameter combinations that contain time to maturity. The figure also shows that the result for the 2 layer model is very similar, but contains some more outliers.
The top result for this validaion uses the following exogenous variables:

- **One Layer:**
- days passed since news
- stock price
- sentiment
- buzz
- **Two Layers:**
- days passed since news
- news flag
- stock price
- sentiment
- maturity

achieving a hit rate of 0.5258298 with a RMSE of 0.4412404 and 0.5283924 with a RMSE of 0.1575366. Note that the relatively poor RMSE value can be explained by the very low number of epochs trained to conduct the validation (5).

4.4 DA-RNN

The next model will take the idea implemented in the LSTM network one step further. In order to acurrately predict longer time series, the Dual-Stage Attention-Based Recurrent Neural Network introduced in [14] is split into two parts. First an input attention mechanism selects the relevant driving series, then a temporal attention mechanism selects relevant states across the time steps of our time horizon P. For a more detailed description please refer to [14]
For this section let us select a subseries of length (P-1) of our input time series X, which has n components, i. e.: $X = (x^1, x^2, \dots, x^n)$ with $x^k = (x^k_1, x^k_2, \dots x^k_{(P-1)})^\top \in \mathbf{R}^{P-1}$ and $x_t = (x^1_t, x^2_t, \dots, x^n_t) \in \mathbf{R}^n$. Like we did with the LSTM-network, we will iterate through the series, always looking at chunks of size (P-1), predicting the P-th value. This means that we will use $P-1$ values of our n exogenous time series $X^t := (x_{t\text{-}P+1}, \dots, x_{t-1})$ and values of our target series $Y^t := (y_{t\text{-}P+1}, \dots, y_{t-1})$ to predict our target y_t, more precisely, given X^t and Y^t we want to find a non-linear mapping that yields $\hat{y}_t \in \mathbf{R}$ which will be our prediction for y_t.
What follows are the computation steps done to find our forecast value for $t = P$.

THE ENCODER WITH INPUT ATTENTION is the first mechanism our n exogenous datapoints $(x^1_t, x^2_t, \dots, x^n_t)$ will pass through at time step t of a total of *P-1* steps. The aim is to learn a mapping from x_t to h^{enc}_t at time step t, where $h^{enc}_t \in \mathbb{R}^{m^{enc}}$ is the output of the Encoder. The input attention layer is

$$iatt^k_t = v^\top_{iatt} tanh(M_{iatt}(h^{enc}_{t-1}; s^{enc}_{t-1}) + W_{iatt} x^k), 1 \leq k \leq n$$

where $v_{iatt} \in \mathbb{R}^{P-1}$, $M_{iatt} \in \mathbb{R}^{(P-1)\times 2m^{enc}}$ and $W_{iatt} \in \mathbb{R}^{(P-1)\times(P-1)}$ are parameters to be learned and s^{enc}_{t-1} is the cell state of the encoder at time $t-1$ as introduced below.
We want the weights to sum to 1, and therefore apply the softmax function to our weights:

$$\alpha_t = \sigma(iatt_t)$$

with $\alpha_t \in \mathbb{R}^n$. The weight α_t measures the importance of the k-th exogenous series at time t. In other words, it determines the amount of attention the various driving series get at time t.
Our weight adjusted input series

$$\tilde{x}_t = (\alpha^1_t x^1_t, \alpha^2_t x^2_t, \dots, \alpha^n_t x^n_t)$$

will be now put into the Encoder LSTM unit f^{enc} as defined below to learn the mapping:

$$h^{enc}_t = f^{enc}(h^{enc}_{t-1}, \tilde{x}_t),$$

yielding our new output h_t^{enc}, that will be forwarded to the temporal attention layer, which we discuss later.

THE ENCODER will be an LSTM unit with forget gate (a gated recurrent unit (GRU) would be possible aswell), that maps the weigthed input $\tilde{x}_t \in \mathbb{R}^n$ and the last encoder output $h_{t-1}^{enc} \in \mathbb{R}^{m^{enc}}$ to the new encoder output:

$$h_t^{enc} = f^{enc}(h_{t-1}^{enc}, \tilde{x}_t).$$

The equations for the encoder unit are as follows:

$$f_t^{enc} = \sigma(M_f^{enc}(h_{t-1}^{enc}; \tilde{x}_t) + b_f^{enc}) \tag{4.3}$$
$$i_t^{enc} = \sigma(M_i^{enc}(h_{t-1}^{enc}; \tilde{x}_t) + b_i^{enc}) \tag{4.4}$$
$$o_t^{enc} = \sigma(M_o^{enc}(h_{t-1}^{enc}; \tilde{x}_t) + b_o^{enc}) \tag{4.5}$$
$$s_t^{enc} = f_t \odot s_{t-1}^{enc} + i_t^{enc} \odot tanh(M_s^{enc}(h_{t-1}^{enc}; \tilde{x}_t) + b_s^{enc}) \tag{4.6}$$
$$h_t^{enc} = o_t^{enc} \odot tanh(s_t^{enc}) \tag{4.7}$$

where σ is the sigmoid function (see:2.14), $\odot$ is the elementwise multiplication operator and $(h_{t-1}^{enc}; \tilde{x}_t)$ is the concatenation of h_{t-1}^{enc} and $\tilde{x}_t$. The following parameters have to be learned:
$M_f^{enc},\ M_i^{enc}\ M_o^{enc},\ M_s^{enc} \in \mathbb{R}^{m^{enc} \times (m^{enc}+n)}$, $b_f^{enc},\ b_i^{enc},\ b_o^{enc},\ b_s^{enc} \in \mathbb{R}^{m^{enc}}$.

THE TEMPORAL ATTENTION LAYER takes as input the outputs of the encoder $h_k^{enc}\ k = 1, \ldots, P$ and the previous output h_{t-1}^{dec} and cell state s_{t-1}^{dec} of the decoder, which will be discussed below. It generates attention weights, using, like the input attention layer, a perceptron:

$$tatt_t^k = v_{tatt} tanh(M_{tatt}(h_{t-1}^{dec}; s_{t-1}^{dec}) + W_{tatt} h_k^{enc}), \qquad 1 \leq k \leq P-1$$

where $(h_{t-1}^{dec}; s_{t-1}^{dec}) \in \mathbb{R}^{2m^{dec}}$ and the parameters to learn are: $v_{tatt} \in \mathbb{R}^{m^{enc}}$, $W_{tatt} \in \mathbb{R}^{m^{enc} \times m^{enc}}$, and $M_{tatt} \in \mathbb{R}^{m^{enc} \times 2m^{dec}}$. Again the softmax function is applied in order to get weights that sum to 1:

$$\beta_t = \sigma(tatt_t)$$

As β_t^k measures the importance of the k-th encoder output, we compute the weighted sum of the encoder outputs, called the **context vector** as:

$$c_t = \sum_{k=1}^{P} \beta_t^k h_k^{enc}.$$

Combining the given target series $(y_1, y_2, \ldots, y_{P-1})$ with c_{t-1}:

$$\tilde{y}_{t-1} = M(y_{t-1}, c_{t-1}) + b$$

where $M \in \mathbb{R}^{m^{enc}+1}$ and $b \in \mathbb{R}$ have to be learned, we get $\tilde{y}_{t-1}$ which will be fed to the decoder LSTM unit:

$$h_t^{dec} = f^{enc}(h_{t-1}^{dec}, \tilde{y}_{t-1})$$

THE DECODER , just like the encoder, will be a LSTM unit. This choice is made because LSTM networks can pick up long term dependencies. The decoder is determined through the following equalities:

$$f_t^{dec} = \sigma(M_f^{dec}(h_{t-1}^{dec}; \tilde{y}_t) + b_f^{dec}) \tag{4.8}$$
$$i_t^{dec} = \sigma(M_i^{dec}(h_{t-1}^{dec}; \tilde{y}_t) + b_i^{dec}) \tag{4.9}$$
$$o_t^{dec} = \sigma(M_o^{dec}(h_{t-1}^{dec}; \tilde{y}_t) + b_o^{dec}) \tag{4.10}$$
$$s_t^{dec} = f_t^{dec} \odot s_{t-1}^{dec} + i_t^{dec} \odot tanh(M_s^{dec}(h_{t-1}^{dec}; \tilde{y}_t) + b_s^{dec}) \tag{4.11}$$
$$h_t^{dec} = o_t^{dec} \odot tanh(s_t^{dec}) \tag{4.12}$$

where $M_f^{dec}, M_i^{dec}, M_o^{dec}, M_s^{dec} \in \mathbb{R}^{m^{dec} \times (m^{dec}+1)}$, $b_f^{dec}, b_i^{dec}, b_o^{dec}, b_s^{dec} \in \mathbb{R}^{m^{dec}}$ again are the parameters to be learned.

THE FINAL PREDICTION is made after P steps based on the context vector c_P and the decoder's last output h_P^{dec} via:

$$\hat{y}_P = F(y_1, y_2, \ldots, y_{P-1}, x_1, x_2, \ldots, x_{P-1}) = v(M(h_P^{dec}; c_P) + b_1) + b_2$$

with $M \in \mathbb{R}^{m^{dec} \times (m^{dec}+m^{enc})}$, $b_1 \in \mathbb{R}^{m^{dec}}$, $v \in \mathbb{R}^{m^{dec}}$, $b_2 \in \mathbb{R}$. This gives us the final p rediction result.

THE IMPLEMENTATION of the DA-RNN was also done in the Pytorch environment of Python.
In [14] it is suggested to use minibatch stochastic gradient descent (SGD), together with the Adam optimizer [8] 2.1. As the objective function we chose the mean squared error:

$$\frac{1}{|N|} \sum_{t \in N} (\hat{y}_t - y_t)^2,$$

with N being the indices of our random batch, $|N|$ being the batch size, y_t being the observed value and $\hat{y}_t$ the predicted value, that we obtain when iterating through our input series of length T, by choosing a new set of $2(P-1)$ input values $(Y^t; X^t) = (y_{t\text{-}P+1}, \ldots, y_{t-1}, x_{t\text{-}P+1}, \ldots, x_{t-1})$ to train our DARNN with, every iteration step.

Now that the theoretical steps are clear, we have to choose the following hyperparemeters for our model:

1. learning rate α
2. decay rates for the Adam optimizer β_1, β_2
3. number of epochs
4. batch size
5. hidden size encoder m^{enc}
6. hidden size decoder m^{dec}
7. time horizon P
8. set of exogenous variables

As we did in the case of the LSTM network, we will use all available exogenous variables as input to our DA-RNN network to select the hyperparameters $1 - 7$. After choosing the hyperparameters $1 - 7$ we will determine the set of exogenous variables that delivers the best predictive power. To do that, we will use the predetermined hyperparameters $1 - 7$ and different combinations of exogenous variables in order to make predictions with our DA-RNN model. For every of the possible combinations of exogenous variables we will create a set of predictions. The set of explanatory variables that was used to make the set of predictions having the best average hit rate will get selected, and will later be used for our final predictions.

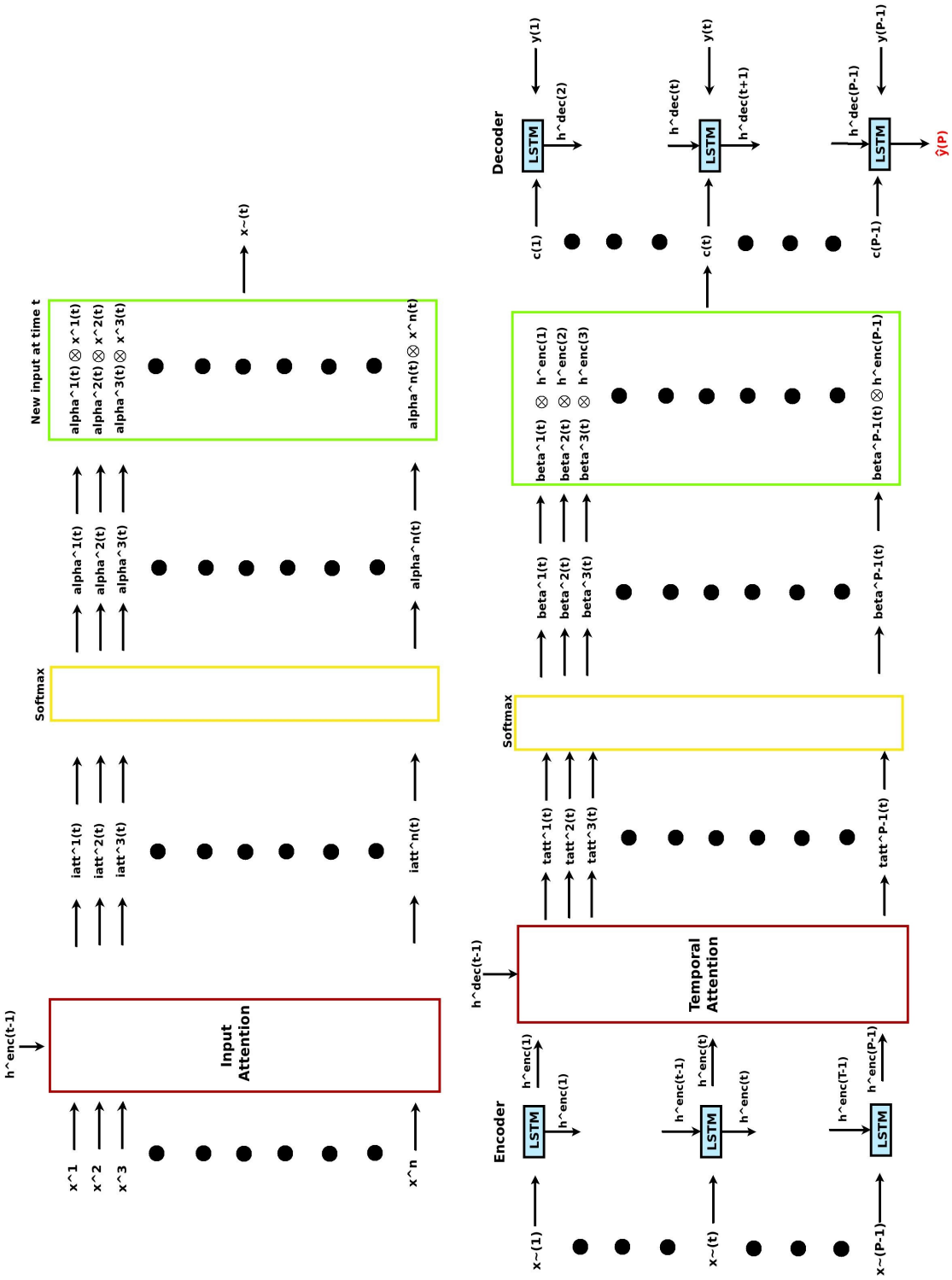

Figure 4.17: DARNN network architecture as depicted in [14]

Very good results could be achieved with a learningrate of 0.001, the same as proposed by [14]. For the decay rate we did adopt the widely used choice, also promoted by the paper: $\beta_1 = 0.9, \beta_2 = 0.999$.
Having chosen these parameters, the remaining were determined by validation, as we did with the LSTM network. Validation was performed by predetermining a certain range of possible hyperparameters and subsequently comparing the hit rate, achieved by those parameters. The parameters were chosen from the following values: hidden-size-encoder=hidden-size-decoder $\in \{16, 32, 64, 128\}$, batch size $\in \{16, 32, 64\}$, time horizon $\in \{8, 16, 32\}$. Again we used all avaliable exogenous variables to create the forecasts for the validation. The results of this procedure can be found in Table 4.26.

Having only limited computational power at hand, we did choose to train for 10 epochs for each set of parameters.
Figure 4.18 shows the superiority of batch size 32 and time horizon 32. For the size of the hidden layer, 64 seems to be favourable over 32, having less extreme outliers. Still hidden size 32 performs very well. The scatterplot of 4.18 shows 3 hyperparameter combinations with particularly low RMSE and hit rate. Those three settings all have batch size = 64, encoder-hidden size = decoder-hidden size = 16 with different time horizons. We therefore want to avoid this combination when forecasting.
Overall this reaffirms our initial parameter choice batch size = 32, hidden size = 32, time horizon = 32.

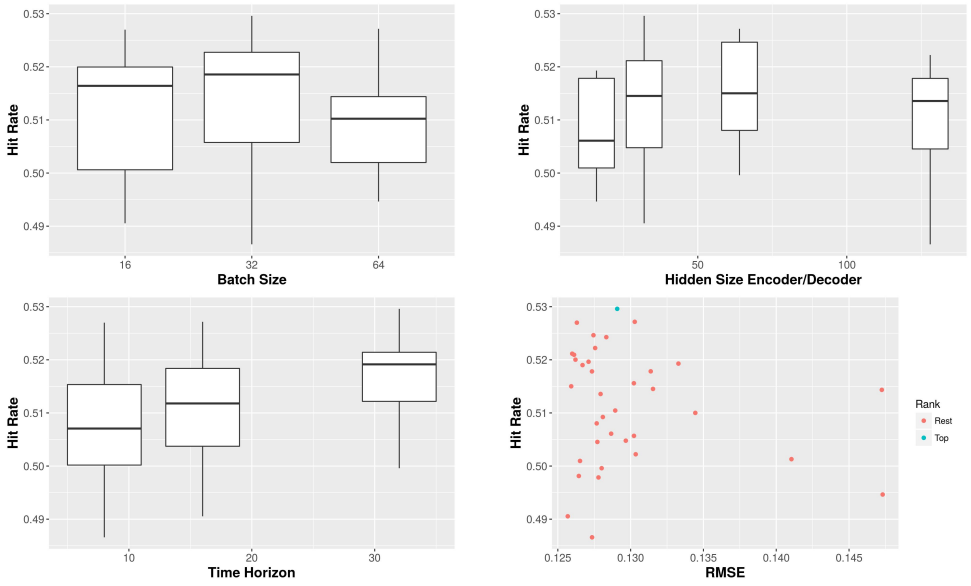

Figure 4.18: Plots for the different hyperparameters of the DARNN network

BATCH SIZE	ENC. HIDDEN	DEC. HIDDEN	P	**hit rate**	RMSE
16	128	128	16	0.5092329	0.12809159
16	128	128	32	0.5196336	0.12710984
16	128	128	8	0.5178164	0.12734542
16	16	16	16	0.4981329	0.12644286
16	16	16	32	0.5190069	0.12669691
16	16	16	8	0.5009622	0.12652369
16	32	32	16	0.4905435	0.12568201
16	32	32	32	0.5211490	0.12598749
16	32	32	8	0.5209427	0.12611606
16	64	64	16	0.5150160	0.12591947
16	64	64	32	0.4995970	0.12800953
16	64	64	8	0.5269891	0.12630757
32	128	128	16	0.5045390	0.12771633
32	128	128	32	0.5222186	0.12756327
32	128	128	8	0.4865873	0.12735061
32	16	16	16	0.5178320	0.13138074
32	16	16	32	0.5192870	0.13328291
32	16	16	8	0.5060917	0.12866176
32	32	32	16	0.5242650	0.12832887
32	32	32	32	0.5295936	0.12907853
32	32	32	8	0.5047755	0.12966824
32	64	64	16	0.5200272	0.12621104
32	64	64	32	0.5246299	0.12744733
32	64	64	8	0.5080341	0.12766874
64	128	128	16	0.5135664	0.12793636
64	128	128	32	0.5155894	0.13022309
64	128	128	8	0.4978564	0.12779212
64	16	16	16	0.5012983	0.14104587
64	16	16	32	0.5143426	0.14725137
64	16	16	8	0.4946430	0.14730916
64	32	32	16	0.5100088	0.13444198
64	32	32	32	0.5022187	0.13035002
64	32	32	8	0.5145185	0.13153710
64	64	64	16	0.5271541	0.13028287
64	64	64	32	0.5056857	0.13022848
64	64	64	8	0.5104524	0.12894283

Table 4.26: Results for the different parameter combinations for the DARNN network

As we did with the ARMA-GARCH model, we now consider groups of bond issued by the same company and choose hyperparameters seperately for every group. Table 4.28 shows the best average hit rate achieved for certain parameter choices. It is evident that Societe Generale Group stands out. This is not just an outlier, but a clear pattern, as the top 15 parameter choices for this company perform better than any other top choice of any other company. At the same time, certain parameter choices result in hit rates being far worse than any of the hit rates achieved by other companies. Again, we observe that the hit rate improves dramatically, when grouping the bond by the issuer, as we calculate the average hit rate of a smaller set of bonds

BATCH	ENC. HIDDEN	DEC. HIDDEN	P	**hit rate**	**company**
16	128	128	32	0.70897	Societe Generale
32	16	16	8	0.54514	Adidas
64	64	64	16	0.53791	Deutsche Bank
64	32	32	16	0.53514	Intesa Sanpaolo
16	32	32	32	0.53301	BNP Paribas

Table 4.28: Top Hyperparameter Choices per Company

The last step is to select the best performing set of hyperparameters for every single bond. Figure 4.19 shows that a lower batch size is generally preferred. A bigger hidden size, on average also results in higher hit rates, which a surprisingly is not the case for our time horizon P. Again, this result does not rule out the possibility of getting better average hit rates with a greater time horizon P. As the computational cost rises strongly when increasing P, we might just not have trained enough epochs to see this effect. Note however that those results resemble those of the LSTM network above.

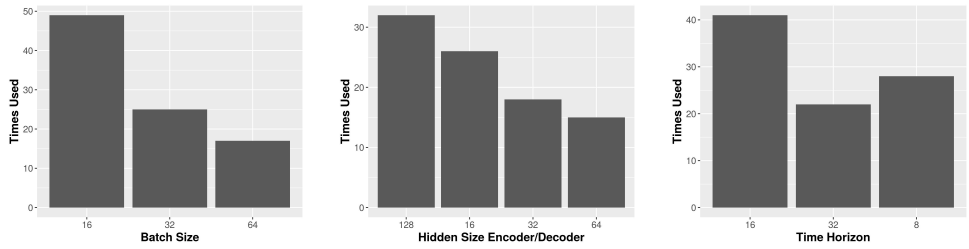

Figure 4.19: Number of times certain hyperparameters were chosen

Before running the forecast, we want to have another look at the given explanatory variables. In chapter 1 we analyzed their correlation, stationarity and seasonality, but were not able to determine their predictive power in a non-linear model. This is exactely what we will look at now. By apply-

ing the model to all possible permutations of our exogenous variables, and then checking the resulting hit rate, we will identify their contributions to our forecasts.
The model with the highest achieved hit rate, used the following hyperparameters:

- news (flag)
- stock
- sentiment
- maturity

and scored a hit rate of 0.5227508 with a RMSE of 0.1540045.
Figure 4.20 plots the hit rates for sets of hyperparameters, all of them containing the indicated varaible to be found on the y-axis. We can see that the time to maturity performed the worst with the stock price being the most important contributor to our forecast result. We however shouldn't rely solely on this result, as there could be favourable combinations including variables that scored low on this boxplot.

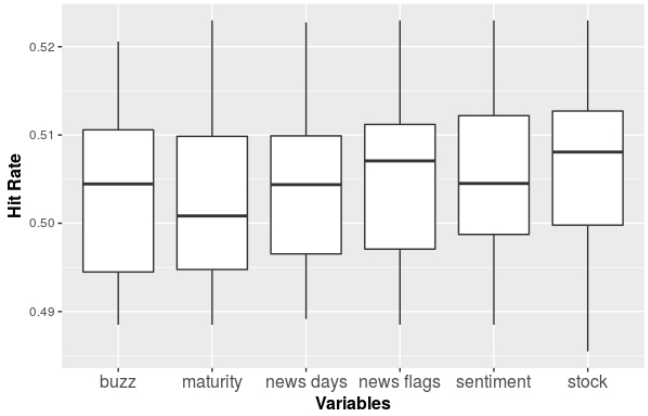

Figure 4.20: Boxplot of different hyperparameters and the achieved hit rate

Keeping in mind, that we have two variables in the dataset that are related to the news events, one counting the days passed and one marking the news report dates with a 1 entry. This means using them both for prediction may not be necessary or will even result in less predictive power. Figure 4.21 (left) shows the resulting hit rate and RMSE when using all possible combinations of explanatory variables conatining the flag variable and the "days passed" variable. We can observe that there are no noticeable patterns highlighting differences between using either of them, apart from the best result for the flag variable being slightly better. Interestingly the best and the second best result uses not only "dayspassed" instead of the flag variable, but also does not make use of the variable containing the stock prices, in contradiction to the interpretation of the boxplot.

Another pair of variables should be taken care of, namely the sentiment and the buzz. Showing slight correlation, as seen in Chapter 1, we expect no additional benefit from using both of them. Figure 4.21 (right) shows, that using the buzz, we can expect a larger varaince of the RMSE, whereas the sentiment delivers the top two results, encouraging the use of the sentiment as an explanatory variable.

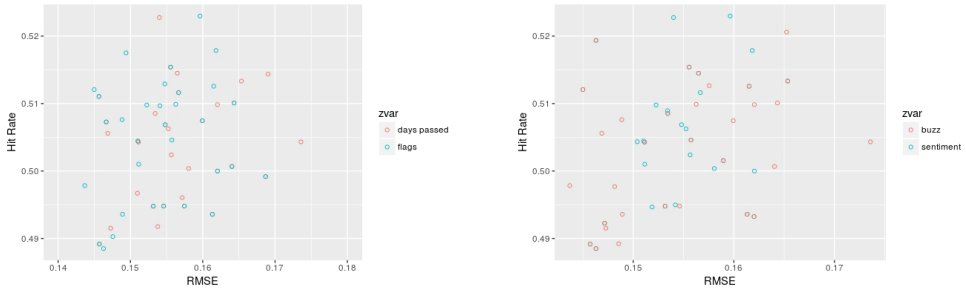

Figure 4.21: Left: Results when using all possible combinations of hyperparameters, including the different flag prameters. Right: Results for all possible combinations, first including buzz, then the sentiment.

Part III

RESULTS

5

RESULTS

Being now acquainted with the different modelling approaches and having set the hyperparameters, we will now take a look at the results. As mentioned in Chapter 3, our final results are the forecasts, generated after training the respective models on 80% of the available data, and then generating 1-step forecasts for the remaining 20% of the data.

5.1 ARMA-GARCH MODEL

Validation showed that when choosing a single parameter set for all the bonds, and when grouping them company-wise, the model including the exogenous parameter in the variance did perform subtly better. That is why we will use this method for our final forecasts. We would expect the other variant of the ARMA-GARCH model to yield highly similar results, as was evident during parameter selection.
Of the two exogenous variables we chose the sentiment, as it did lead us to slightly better results, but again note that the news variable would probably lead to very similar results.

ONE-PARAMETER SET: Generating forecasts with the exogenous variable sentiment, based on the set of hyperparameters for all our 91 bonds, as chosen in Chapter 4 ($(P, Q, p, q) = (4, 5, 2, 3)$), we get an average hit rate of 0.54906 with an average RMSE of 0.20707. This is an exceptional good result. What is even more remarkable, is that the forecasts of ARMA-GARCH models are themselves linear. In fact they are ARMA models, which differ from standard ARMA forecasts only in the computation of the parameters.

COMPANY-WISE GROUPING: This setting quite surprisingly achieved an even higher average hit rate of 0.55166 at an RMSE of 0.18664. Figure 5.1 highlights the very small amount of variance in the hit rate of all three different approaches.

SEPERATE TRAINING: Again we now consider the results, when forecasting every bond with its own parameter set. This resulted in an average hit rate of 0.54989 with an average RMSE of 0.1953974, indicating that choosing individual parameters for every set, does not lead to any extra predictive power in this case. Figure 5.1 illustrates all three results, and highlights their similarity.

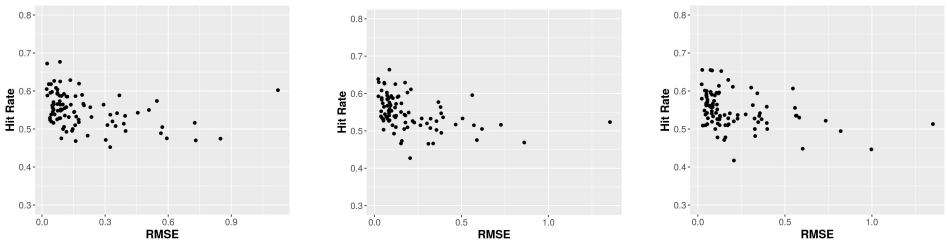

Figure 5.1: Hit Rate plotted against the RMSE for the three different approaches "individual parameter", "company parameter" and "one parameter" (l.t.r.)

5.2 STAR MODEL

ONE-PARAMETER SET: The forecasts with the exogenous variable sentiment, based on the set of hyperparameters for all our 91 bonds, as chosen in Chapter 4 ($(mH, mL) = (5, 6)$), we get an average hit rate of 0.53399 with an average RMSE of 0.18934. This is the worst result until now, but still clearly beats a random draw from $[0, 1]$. Remember that given the threshold variable Z_t, the forecasts inside the continuously many regimes are still a linear function of the current and past values of the time series. A property the two neural network models do not possess.

COMPANY-WISE GROUPING: In contrast to the good result obtained during validation, this approach yields a much lower hit rate of 0.53297 at an RMSE of 0.18978. Note that this hit rate is slightly below the one observed with the one-parameter approach.

SEPERATE TRAINING: Considering the results when forecasting every bond with its own parameter set, we get a very similar average hit rate of 0.53324 with an average RMSE of 0.18917, indicating that choosing individual parameters for every set, does not lead to any extra predictive power in this case, but even leads to worse hit rates compared to the one-parameter approach. Figure 5.2 illustrates all three results, and again highlights their similarity.

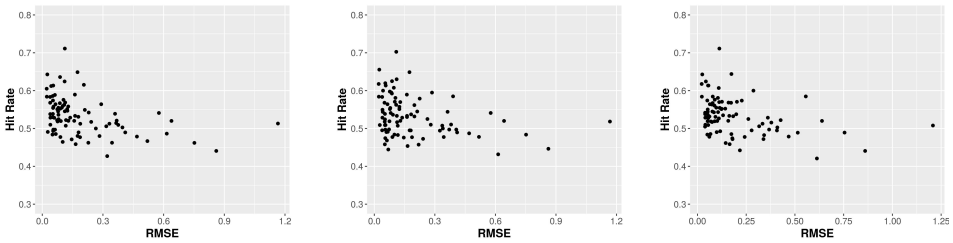

Figure 5.2: Hit Rate plotted against the RMSE for the three different approaches "individual parameter", "company parameter" and "one parameter" (l.t.r.)

5.3 LSTM NETWORK

ONE-PARAMETER SET: We will begin this section with the results we anticipate to be the worst, namely those with one set of hyperparameters for all data sets.
In the previous chapter we did argue why the choice: batch size = 64, hidden size = 16, P = 8 with 2 hidden layers and the exogenous variables "days passed since news", "news flag", "stock price", "sentiment" and "maturity", is reasonable. We did train this LSTM network for 200 epochs, and got a hit rate of 0.519609 with an RMSE of 0.28174. This is better than a random draw from $[0, 1]$ and therefore a satisfying result.
Remember, that when we decided to prefer 2 hidden layers over 1 during validation, the differences in the outcomes of those two models were very minor. Several hyperparameter choices for the 2 layer model performed only slightly, if at all better than the 1 layer counterpart. This is why we did also train a 1 layer model with the same set of hyperparameters and explanatory variables to compare the results. In accordance to the validation findings, assuming only insignificant advantage of the 2 layer model, we got a hit rate of 0.51966 with an RMSE of 0.26102, even very slightly outperforming the 2 layer model.

COMPANY-WISE GROUPING: Proceeding with grouping those bonds, issued by the same company, and training them with the same parameters, we get a hit rate of 0.51395 with a RMSE of 0.25695. While the RMSE decreased, also the hit rate did so, making this model perform worse than the previous one. As supported by the next paragraph, this model is not only a theoretical compromise, but also its' outcomes lie in the middle of our three approaches.

SEPERATE TRAINING: At Validation, the model with separate hyperparameters for every bond, performed by far the best, concerning both hit rate and RMSE. Predicting out of sample however, exposed the weakness of this approach. While during training we could achieve stellar results, the out of sample forecasts were the worst of the three approaches undertaken, with a hit rate of 0.50909 and a RMSE of 0.27949.
See Figure 5.3 and note that the variation in the hit rate decreases very slightly, when only one parameter set is used for all bonds.
It has to be noted that this model resulted in memory shortage for some big datasets (24, 29, 35 , 40, 47,60, 63, 64, 71, 81, 90), in turn resulting to a calculation abortion. Those datasets have been trained on a reduced training sample that has been cropped, removing the first 20% of its entries. This was done for all three approaches, thus did not result in any bias. The problem could probably have been avoided using a stronger, more recent computer.

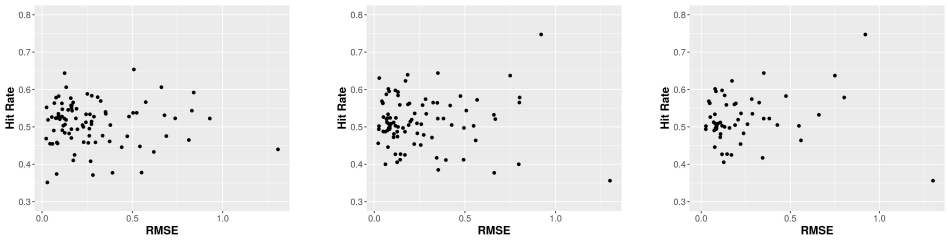

Figure 5.3: Hit Rate plotted against the RMSE for the three different approaches "individual parameter", "company parameter" and "one parameter" (l.t.r.)

5.4 DARNN NETWORK

ONE-PARAMETER SET: The more complex structure of this model, seems to bear fruit. Compared to the same parameter approach but using the LSTM model, the hit rate 0.533411 with the RMSE 0.16726 constitutes a strong improvement. The time needed to train the two models was roughly the same. On the machine available to train this model however, we got the same error as we did when training the LSTM model, alluding insufficient memory. As with the LSTM model, we did trim the training set of those bonds, again for all three approaches.

COMPANY-WISE GROUPING: A still very good hit rate of 0.515637 with a RMSE of 0.20301 was observed when using hyperparameters per group. Strongly resembling the outcome of the LSTM network, we again observe an impairing hit rate, the less grouping and thus averaging takes place, when choosing our hyperparameters.

SEPERATE TRAINING: It seems as the DARNN behaves very similar to the LSTM nework, only on a slightly higher level of hit rates. For training our model with separate parameters for every dataset we could score a hit rate of 0.515169 and a RMSE of 0.20426
Note that the same behaviour as that of the LSTM model, can be observed for the hit rates. When looking at Figure 5.4, we can see a decreasing of the hit rate's varaince for the one-parameter approach.

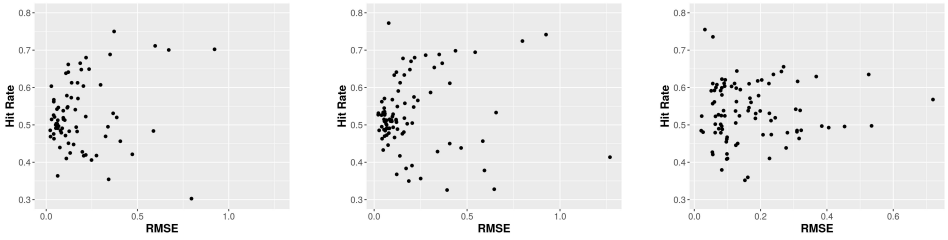

Figure 5.4: Hit Rate plotted against the RMSE for the three different approaches "individual parameter", "company parameter" and "one parameter" (l.t.r.)

5.5 RESIDUAL DIAGNOSTICS:

A Residual e_t in forecasting is the difference between the one-step prediction and the observed value at that time $e_t = y_t - \hat{y}_t$. Residuals should have the following two properties:

- The residuals are **uncorrelated**
- The residuals have **zero mean**

If there is correlation between residuals, then there is information left in them, which should have been used for forecasting. In case the residuals do not have zero mean, then the forecasts are biased. In any of these two cases, the forecasting method can be improved to produce more accurate predictions.
We will analyse the residuals coming from the best performing sub method of each method, (for these sub methods see: Table 5.2), and check them for bias and correlation. This analysis is done separately for each bond in the dataset. In order to check for correlation, the Ljung-Box test was performed. If there was evidence for correlation, the autocorrelation function was inspected to identify possible lags at which correlation occurs.

BIAS was very minor if existent. For all four methods, there were no bonds for which the mean of the residuals was greater than 0.1, and there was a maximum of two bonds per method that had a mean of residuals between 0.1 and 0.05. Furthermore the mean was in some cases positive and in some cases negative, with no conspicuous patterns observable.

CORRELATION was observable to a different extent for the four methods. The residuals of the STAR model were the least correlated, with 62 bonds yielding p-values of above 0.05, which let's us conclude that the residuals for those bonds are not distinguishable from a white noise process. The remaining bonds for this models, showed relevant correlation at lag 19. This means that the STAR model did not make use of all the information provided. Lag 19 indicates that the influence of the job market data wasn't fully covered by the model's predictions.
For the ARMA-GARCH model, there were 59 bonds with p-values above 0.05. The correlation of the remaining residuals occur at lag 19 and in two cases at lag 3.
The residuals of the LSTM model behave different. Not only are there more bonds where the residuals show correlation (50), but the correlations occur at different lags, with no lags standing out strongly. Only at lag 19 correlations occur a bit more frequently than at other time lags.
The DARNN surprisingly performed worst with 53 bonds with p-values below 0.05, meaning that for only 38 bonds the 0-hypothesis of independence wasn't rejected. As was the case with the LSTM's residuals, correlation occurs at different lags, and correlations at lag 19 were a little more frequent.

The varying residual behaviour of the two neural networks suggests that they have for various bonds picked up different dependencies, that were not actually present in the spread change time series. This might be the result of the optimization algorithm reaching saddle points. As we used the mini-batch gradient descent (Chapter 2), the chances of getting trapped at such saddle points are elevated. With the computational power at hand, reducing the batch size however makes computation in reasonable time, impossible .
The correlations at lag 19 on the other hand show that the information the news variable delivered, wasn't fully exploited in all of the models. This is no surprise for the ARMA-GARCH and the STAR model, as the news variable wasn't part of the input for those models, but the LSTM and the DARNN theoretically could have used this information.

5.6 DIEBOLD MARIANO TEST:

Having analysed the two neural network models on their own, we now want to apply the Diebold Mariano test Chapter 3, to see whether the difference in the predictive power between the four models, is significant. We will apply the test to the results coming from the one-parameter approach, as it did perform the best overall.
Lets keep in mind that the 0-hypothesis of the DM test is, that the two methods have the same forecast accuracy. Out of curiosity, we also compare the other 2 approaches' results for the LSTM and the DARNN model.

LSTM – DARNN First we did apply the test to the results coming from individual parameters. This resulted in 73 bonds with a p-value of under 0.05 indicating that the 0-hypotheses of equal predictive power gets rejected in those cases.
Next the DM test was executed on the DARNN and the LSTM model, resulting from grouping the bonds. There were 70 bonds with a p-value smaller than our 0.05 threshold. Conducting the same procedure on the results coming form the model with only one parameter we observed 67 bonds with p-values below the threshold.
To summarize one can say, the more grouping is done concerning the choice of the hyperparameters, the more similar the predictive power of the two models get. Note that more grouping also results in better overall predictive power for both models.
We then tested for the alternative 0-hypotheses that the DARNN forecasts are less accurate than the LSTM forecasts and that the DARNN forecasts are more accurate than the LSTM forecasts, which yielded 66 p-values of under 0.05 and 1 such p-value respectively for results coming from individual parameters. For company-wise grouping we got 66 and 2 small p-values and for the one-parameter approach we got 70 and 7 such values. This suggests that the forecasts coming from the DARNN model are more accurate than

those coming from the LSTM model for all three approaches of choosing the hyperparameter set.

ARMA GARCH – DARNN Comparing the ARMA GARCH model to the DARNN, we get different results, with now only 77 companies with a p-value smaller than 0.05. This means that for about 85% of our bonds, the DARNN and the ARMA-GARCH model deliver significantly different forecasts. That the forecasts of the LSTM and the DARNN have more similar forecast accuracy as the ARMA GARCH and the DARNN seems intuitive, as the DARNN makes use of the LSTM network.
We again tested for alternative 0-hypotheses. Testing the 0-hypothesis: "Forecasts from the ARMA-GARCH model are less accurate than those coming from the DARNN model" yielded 6 p-values of under 0.05 whereas "Forecasts from the ARMA-GARCH model are more accurate than those coming from the DARNN model" yielded 20 such p-values when choosing the one-parameter approach. This result suggests that the DARNN forecasts are better for more bonds, but note that it does this with relatively small significance. Table 5.2 shows the lower RMSE of the DARNN model, which could explain this finding.

ARMA GARCH – LSTM The DM test for the ARMA-GARCH and the LSTM model, resulted in 56 bonds small p-values, making those two models' forecasts' accuracies' similarity the highest detected between all pairs of models.
Testing for the hypothesis that forecasts of the ARMA-GARCH model are worse than those coming from the LSTM model resulted in 59 p-values of under 0.05. Testing for the hypothesis that forecasts of the ARMA-GARCH model are better than those coming from the LSTM model resulted in 5 p-values of under 0.05. This would mean that for most bonds the ARMA-GARCH model's forecasts outperform those of the LSTM model.

STAR – DARNN 57 values under 0.05 were found. This means that the forecasts produced by this two models are approximately as similar as those of the ARMA-GARCH and the LSTM model.
The alternative hypothesis that the STAR model's forecasts are less accurate results in 26 p-values of under 0.05 whereas the hypothesis that the STAR model's forecasts are more accurate yields 35 such p-values. Thus the DARNN model's forecasts seem to have better accuracy in some cases, but for the majority of bonds no clear statement on better or worse accuracy of either of the two models can be made.

STAR – LSTM 59 values under 0.05 were found. This means that the results of the DM test for these two models are very close to the results for the previous pair of models, namely the DARNN and the STAR model.
There were 36 p-values of under 0.05 for the hypothesis that the STAR

model's forecast are less accurate than the LSTM model's forecasts, and 11 such p-values for the hypothesis that they are achieve greater accuracy.

STAR – ARMA GARCH For the STAR and the ARMA GARCH model, we anticipate highly similar forecasting accuracy, as they both evolved from ARMA models. In fact for 60 bonds, the forecasts yielded a p-value smaller than 0.05. Thus about 34% of the bonds delivered similar forecasts.
The tests for the alternative hypotheses that the STAR model's forecasts have better accuracy resulted in 10 p-values of under 0.05 whereas the test that the STAR model's forecasts are less accurate resulted in 9 such p-values. This result is the least significant. It shows that for the majority of bonds, non of these two models is clearly better than the other.

5.7 SUMMARY

The best results of each method are summarized in Table 5.2 and Figure 5.5.

MODEL	SUB-MODEL	RMSE	**hit rate**
ARMA-GARCH	company-wise	0.18664	0.55166
STAR	one-parameter	0.18934	0.53399
DARNN	one-parameter	0.16726	0.533411
LSTM	one-parameter	0.28174	0.519609

Table 5.2: Summary of the results

With the DARNN, our most sophisticated and just recently developed model on the third place concerning the hit rate, a lot of questions arise. Especially given the success of this model when applied to different machine learning tasks. Regarding the RMSE however, the DARNN model did outperform the other methods with distinctly lower RMSE values.
The results also show the unexpected good forecasting power of the ARMA-GARCH models with their forecasts being linear, once the ARMA parameters are chosen.
The Diebold Mariano test results are summarized in Table 5.4. Note that the two models evolving from AR-processes reject the 0-hypothesis of the same forecast accuracy in 66% of the cases, and the two neural network models even in 74% of the cases.

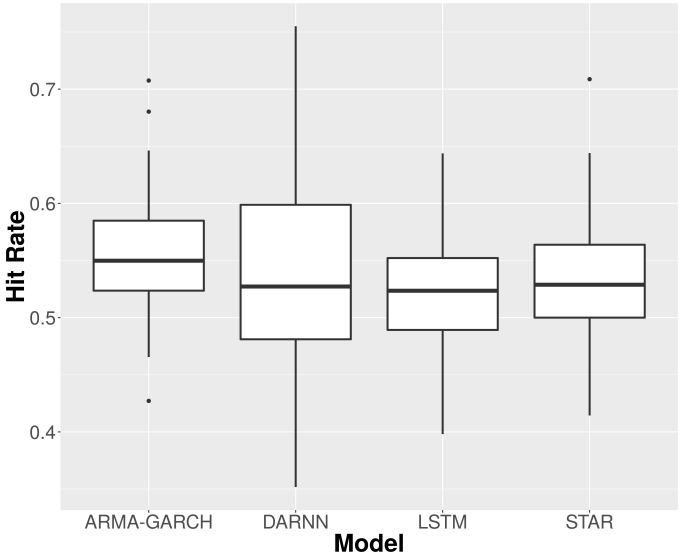

Figure 5.5: Hit rates of the best results of each method

model				
A.-G.	65 62 6	7 85 22	11 66 10	0
STAR	40 65 12	29 63 38	0	10 66 11
DARNN	76 74 8	0	38 63 29	22 85 7
LSTM	0	8 74 76	12 65 40	6 62 65
	LSTM	DARNN	STAR	A.-G.

Table 5.4: Summary of the DM test results, depicting for every pair of modles the % of cases that we rejected the 0-hypothesis of less, equal and greater predictive power of the model in the regarding row to the model in the regarding column

6 CONCLUSION

6.1 RESULTS

After trying different approaches with different data, one is inclined to believe that the price changes in the bond market remain almost unpredictable. With all of our models performing better than a random draw from $[0, 1]$, it should however still be possible to beat the market, by implementing trading strategies that exploit this extra knowledge.
Nevertheless the significance of the results should not be overestimated, given the relative small amount of bond data used to perform the forecasts. With bonds issued by companies from a wider variety of markets, the result may have been slightly different, due to possible correlations to other exogenous variables, not detected with our data sets, or due to the limited computational power available.
The lack of computational power did not so much affect the ARMA-GARCH and the STAR model's forecasts, as the convergence criteria were reached there, giving fairly good results, but did probably affect the two artificial neural networks which were trained a maximum of 10 hours per set of 91 bonds. Apart from the limited time the models were trained and limitations concerning the choice of the hyperparameters, we saw that for some bonds and hyperparameters we ran out of memory, making the desired calculations impossible altogether. The rise of machine learning does not by chance coincide with the rapid development of CPU and GPU speeds. Usual settings for working with LSTM networks involve a cluster with multiple cores, that are training the model for several hours.
But we could also observe, that given those limited resources, the ANN models, are no guarantee for better results, putting into perspective the demand fore extra computational force.
We should also keep in mind that the list of ANN methods used in this thesis is by no means exhaustive and that there is a multitude of possible variants of the ANN models used.

6.2 PROSPECTIVE

For future work on the prediction of bond prices, modelling the information contained in earnings releases and job market reports, could provide valuable information. Furthermore including the release dates of other macroeconomic data provided by central banks and in a second step modelling the data released at this dates, could lead to even higher prediction accuracy.

Talking about improved predictive power, we should keep in mind that there are a lot of highly random factors that have not been considered in this thesis but influence the market greatly. The recent U.S. election for example exposed the major influence its outcome has on markets, and at the same time showed how hard it is to predict this outcome. Some might even render such political events unpredictable but the rise of social media, and the digitalisation of our communication in general, might change this in the near future, providing an even broader and diverse range of data that can potentially be exploited for modelling the bond market.

Part IV

APPENDIX

A

APPENDIX

NUMBER	ISIN	COMPANY NAME	SIZE
1	DE000A1ML0D9	Adidas AG	1172
2	DE000DB2KU80	Deutsche Bank AG	845
3	DE000DB2KUU4	Deutsche Bank AG	829
4	DE000DB2KX53	Deutsche Bank AG	801
5	DE000DB2KX61	Deutsche Bank AG	802
6	DE000DB2KXL7	Deutsche Bank AG	970
7	DE000DB2KYG5	Deutsche Bank AG	804
8	DE000DB5DBR8	Deutsche Bank AG	924
9	DE000DB5DCV8	Deutsche Bank AG	871
10	DE000DB5DDN3	Deutsche Bank AG	815
11	DE000DB5DDP8	Deutsche Bank AG	805
12	DE000DB5DDQ6	Deutsche Bank AG	815
13	DE000DB5DDT0	Deutsche Bank AG	800
14	DE000DB5DEF7	Deutsche Bank AG	955
15	DE000DB5DEK7	Deutsche Bank AG	978
16	DE000DB5DFA5	Deutsche Bank AG	984
17	DE000DB5DFB3	Deutsche Bank AG	848
18	DE000DB7UP56	Deutsche Bank AG	992
19	DE000DB7UQ22	Deutsche Bank AG	832
20	DE000DB7UQT2	Deutsche Bank AG	821
21	DE000DB7URS2	Deutsche Bank AG	841
22	DE000DB7XLA7	Deutsche Bank AG	828
23	DE000DB7XMJ6	Deutsche Bank AG	808
24	DE000DB7XMQ1	Deutsche Bank AG	810
25	DE000DB7XMZ2	Deutsche Bank AG	868
26	DE000DB7XNB1	Deutsche Bank AG	915
27	DE000DB7XNC9	Deutsche Bank AG	889
28	DE000DB7XNM8	Deutsche Bank AG	923
29	DE000DB7XNR7	Deutsche Bank AG	952
30	DE000DB7XNS5	Deutsche Bank AG	960
31	DE000DB7XNT3	Deutsche Bank AG	961

NUMBER	ISIN	COMPANY NAME	SIZE
32	DE000DB7XNW7	Deutsche Bank AG	994
33	DE000DB7XNX5	Deutsche Bank AG	967
34	DE000DB7XNZ0	Deutsche Bank AG	998
35	DE000DB7XPA8	Deutsche Bank AG	1001
36	DE000DB7XPB6	Deutsche Bank AG	802
37	DE000DB7XPC4	Deutsche Bank AG	880
38	DE000DB7XPD2	Deutsche Bank AG	1001
39	DE000DB7XPE0	Deutsche Bank AG	1001
40	DE000DB7XPF7	Deutsche Bank AG	827
41	DE000DB7XPL5	Deutsche Bank AG	996
42	DE000DB7XPU6	Deutsche Bank AG	891
43	DE000DB9ZDH0	Deutsche Bank AG	1040
44	DE000DB9ZDL2	Deutsche Bank AG	1041
45	DE000DB9ZDP3	Deutsche Bank AG	803
46	DE000DB9ZDT5	Deutsche Bank AG	821
47	DE000DB9ZEQ9	Deutsche Bank AG	1017
48	DE000DB9ZEU1	Deutsche Bank AG	833
49	DE000DB9ZEZ0	Deutsche Bank AG	843
50	DE000DB9ZFN3	Deutsche Bank AG	847
51	DE000DB9ZGD2	Deutsche Bank AG	863
52	DE000DB9ZGR2	Deutsche Bank AG	865
53	DE000DB9ZHR0	Deutsche Bank AG	889
54	DE000DB9ZHY6	Deutsche Bank AG	875
55	DE000DB9ZJZ9	Deutsche Bank AG	895
56	DE000DB9ZKJ1	Deutsche Bank AG	900
57	DE000DB9ZLE0	Deutsche Bank AG	954
58	DE000DB9ZLQ4	Deutsche Bank AG	964
59	DE000DB9ZLX0	Deutsche Bank AG	976
60	DE000DB9ZMM1	Deutsche Bank AG	995
61	DE000DB9ZMU4	Deutsche Bank AG	994
62	DE000DB9ZNB2	Deutsche Bank AG	1007
63	DE000DB9ZNY4	Deutsche Bank AG	1012
64	DE000DB9ZQA7	Deutsche Bank AG	1027
65	DE000DB9ZSE5	Deutsche Bank AG	1007
66	FR0010782664	Societe Generale Group	805
67	FR0010785543	Societe Generale Group	802
68	FR0010859967	BNP Paribas	1191

NUMBER	ISIN	COMPANY NAME	SIZE
69	FR0010988873	BNP Paribas	1191
70	FR0011056126	Societe Generale Group	807
71	FR0011059930	BNP Paribas	855
72	FR0011075167	BNP Paribas	1056
73	FR0011082676	BNP Paribas	1030
74	FR0011129873	BNP Paribas	1163
75	FR0011137611	BNP Paribas	1191
76	FR0011160779	BNP Paribas	1029
77	FR0011164862	BNP Paribas	1054
78	FR0011203165	BNP Paribas	1014
79	FR0011253665	BNP Paribas	825
80	FR0011417583	BNP Paribas	859
81	FR0011470921	BNP Paribas	899
82	IT0004537251	Intesa Sanpaolo SPA	919
83	IT0004594658	Intesa Sanpaolo SPA	801
84	IT0004695018	Intesa Sanpaolo SPA	899
85	IT0004703952	Intesa Sanpaolo SPA	1050
86	IT0004717333	Intesa Sanpaolo SPA	1050
87	IT0004806615	Intesa Sanpaolo SPA	1035
88	IT0004808710	Intesa Sanpaolo SPA	801
89	IT0004814098	Intesa Sanpaolo SPA	999
90	IT0004814973	Intesa Sanpaolo SPA	814
91	IT0004839251	Intesa Sanpaolo SPA	1029

Table A.1: Information on the datasets used in this thesis

BIBLIOGRAPHY

[1] F. Fallside A. J. Robinson. *The utility driven dynamic error propagation network*. Technical Report CUED/F-INFENG/TR.1. Cambridge University Engineering Department, 1987.

[2] Yoshua Bengio. "Practical recommendations for gradient-based training of deep architectures." In: *CoRR* abs/1206.5533 (2012). URL: http://arxiv.org/abs/1206.5533.

[3] Jean-Michel Zakoıan Christian Francq. *GARCH Models, Structure, Statistical Inference and Financial Applications*. France: John Wiley Sons Ltd, 2010.

[4] Harvey D., Leybourne S., and Newbold P. "Testing the equality of prediction mean squared errors." In: *International Journal of forecasting* 13(2) (1997), pp. 281–291.

[5] Francis X. Diebold and Roberto S. Mariano. "Comparing Predictive Accuracy." In: *Journal of Business and Economic Statistics* 13 (1995), pp. 253–265.

[6] Sepp Hochreiter. *Untersuchunge zu dynamischen neuronalen Netzen*. Diplomarbeit im Fach Informatik. Munich, Germany, 1991.

[7] Michael Fang James Hong. *Sentiment Analysis with Deeply Learned Distributed Representations of Variable Length Texts*. Carlifornia, USA: Stanford University Press.

[8] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization." In: *CoRR* abs/1412.6980 (2014). arXiv: 1412.6980. URL: http://arxiv.org/abs/1412.6980.

[9] T. Subba Raon M. B. Priestley. *A Test for Non-Stationarity of Time-Series*. Vol. 31. Carlifornia, USA: Wiley, 1969, pp. 140–149.

[10] Richard L. Peterson. *Trading on Sentiment*. Finance Series. Carlifornia, USA: Wiley, 2016.

[11] Brian Sack Refet S. Gurkaynak and Jonathan H. Wright. *The U.S. Treasury Yield Curve: 1961 to the Present*. Washington, D.C., USA: Federal Reserve Board, 2006.

[12] Jürgen Schmidhuber Sepp Hochreiter. "Long Short term Memory." In: Neural Computation 9(8) (1997), pp. 1735–1780.

[13] Weijters A.J.M.M. Braspenning P.J. Thuijsman F. *Artificial Neural Networks, An Introduction to ANN Theory and Practice*. Lecture Notes in Computer Sciences. Springer, 1995.

[14] Haifeng Chen Wei Cheng Guofei Jiang Garrison W. Cottrell Yao Qin Dongjin Song. *A Dual-Stage Attention-Based Recurrent Neural Network for Time Series Prediction*. San Diego, USA: University of California San Diego, NEC Laboratories America Inc., 2017.

[15] Pengjie Gao Zhi Da Joseph Engelberg. *The Sum of all FEARS: Investor Sentiment and Asset Prices*. Vol. 28. Oxford University Press, 2015, pp. 1–32.

COLOPHON

This document was typeset using the typographical look-and-feel `classicthesis` developed by André Miede and Ivo Pletikosić.

Final Version as of May 11, 2018 (`classicthesis` version 3.0).