Forecasting cash flow with machine learning techniques

by

Rick van Rijn (SNR: 2026804)

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Business Analytics & Operational Research

Tilburg School of Economics and Management
Tilburg University

Supervised by: Hennie Daniels

Date: 10 November 2024

Chapter 1

Management Summary

This Thesis aimed to determine the most effective cash flow forecasting methods for GS Interfer by comparing various models, including ETS, SARIMA, and LSTM. The forecasts were generated for both weekly and monthly intervals. The results show that the LSTM model outperformed all others regarding RMSE. For monthly forcasts SARIMA was best in terms of MAE. Therefore it is recommended to use LSTM models for forecasting the weekly cashflow. For mothly cashflow it is recommended to use either LSTM or SARIMA. The study was limited to historical data and did not consider external economic factors that may influence future cash flows. Therefore, further research into incorporating external variables into the models is suggested.

Contents

Chapter 2

Introduction

2.1 Research Context and Objectives

Many companies miss significant opportunities by not adequately leveraging data and machine learning techniques. Cash is a critical driver for mid-market companies, as its availability directly impacts prospects. A lack of insight into cash flow can hinder strategic investment decisions, potentially giving competitors an advantage and decreasing market share.

To address this, mid-market companies often rely on outdated tools for cash position insight. This thesis aims to explore time series, artificial intelligence, and machine learning techniques to create an accurate cash flow forecast.

In this thesis, three methodologies are used to construct a forecast. The first method is exponential smoothing. This method uses weighted averages of previous observations to predict future values. It decomposes the time series in an error, trend, and seasonality component to make forecasts. Understanding and modeling these components allow analysts to make accurate predictions and gain valuable insight into understanding patterns in the data. The second method is SARIMA or seasonal auto-regressive integrated moving average. It is the result of combining auto-regressive models and moving average models. SARIMA models use past values and past forecast errors to predict future values. It is important for SARIMA models that the data is stationary. To test this the Kwiatkowski-Phillips-Schmidt-Shin test and the Augmented Dickey-Fuller test are used. The last method is the Long Short-term Memory neural network. This is a variation of a neural network that has an internal memory, which allows it to retain information about previous inputs. This makes them suitable for handling time-dependent data.

2.2 Collaboration with Den of Data and GS Interfer

This thesis was conducted in collaboration with Den of Data, a consultancy that seeks to unlock potential through data-driven strategies. Den of Data assists companies by automating data visibility and optimizing processes, thereby enhancing business outcomes and competitive positioning.

The data used in this research is provided by GS Interfer, a Den of Data client. GS Interfer supplies fasteners, hinges, locks, hardware, and impulse concepts in tools, painting supplies, and personal protective equipment to hardware store chains across the Netherlands and Belgium. The data consists of transaction records from GS Interfer.

2.3 Research Questions and Objectives

The primary objective of this thesis is to determine which of the proposed forecasting methods can most accurately predict cash flow. In addition, the thesis explores whether forecasting the income and expenditure separately improves the forecast. Also, this thesis aims to research whether combining different forecasting models can improve predictive performance. These objectives will be addressed by forecasting the cash flow.

2.4 Thesis Structure

This thesis is organized into six chapters. Chapter three introduces the data that is used to make the forecasts. The data is explained, summarized, and visualized to make it understandable.

Chapter four introduces the methods that will be used to make the forecasts. Each method has a section dedicated to the methodology. This is the part that explains what the method is and how it works. Each method also has a section about parameter tuning. In this part, it is explained how the parameters of the methods are optimized to get feasible forecasts.

In chapter five, the forecasts are made with the methods explained in chapter four. These forecasts will be evaluated on several evaluation methods, which will also be described. All results will be taken into consideration to build the final forecast.

Chapter six will provide the conclusion to the research questions. (more text later)

2.5 Literature Review

Zhang, Tang, Gao, and Pan (2018) explored optimizing a financial trading strategy using LSTM neural networks to forecast stock prices based on data from the SP 500 Index. By employing LSTMs to capture temporal dependencies, they achieved significant improvements over traditional strategies. Their findings underscore LSTM's ability to model complex time dependencies effectively.

Singh and Tripathi (2014) conducted a comparative analysis of machine learning techniques in stock price prediction, using historical stock data. They examined ARIMA, Exponential Smoothing, and neural networks, revealing that ARIMA and Exponential Smoothing excelled in stable markets, whereas neural networks were superior in handling fluctuations, illustrating their adaptability in dynamic environments.

Mishra (2016) compared ARIMA and machine learning methods for forecasting financial time series data, drawing from various market indices. Mishra found ARIMA suitable for stationary data, while neural networks excelled with non-linear patterns. This research highlights how machine learning models can adapt to complexities in financial data where traditional methods might falter.

Fischer and Krauss (2018) assessed machine learning's role in replicating trading strategies, using a hybrid model combining ARIMA and LSTM with historical financial data. They found that the hybrid model replicated the success of the original strategy while adapting to market volatility, suggesting benefits in leveraging both linear and non-linear patterns within financial data.

Brownlee (2019) evaluated deep learning models, particularly LSTMs, for financial time series forecasting across multiple assets. His study demonstrated that LSTM models offered substantial predictive improvements, emphasizing their utility in complex financial forecasting tasks due to their capacity for learning long-term dependencies.

Chapter 3

Data

3.1 Data Source

The dataset used in this study comes from GS Interfer, a supplier based in the Netherlands and Belgium. GS Interfer provides various hardware products, including fasteners, hinges, locks, and tools to hardware store chains. The company also offers impulse concepts in areas such as hand tools, painting supplies, and personal protective equipment. Additionally, GS Interfer serves as a service provider for several A-brands. The data comprises transactions from two key tables: debtor transactions and creditor transactions.

3.2 Data Overview

The data spans the period from 2014 to 2023, with the training data covering the years 2014 to 2022, and the test data for 2023. The transactions in the dataset are aggregated at a weekly and monthly level.

3.3 Data Tables

3.3.1 Debtor Transactions

The debtor transactions table consists of the following columns:

- **Amount**: The monetary amount owed by the debtor.
- **Debtor**: The identifier for the debtor (e.g., the client).
- **Date**: The date the payment is due or made.

3.3.2 Creditor Transactions

The creditor transactions table includes the following columns:

- **Amount**: The monetary amount owed to the creditor.
- **Creditor**: The identifier for the creditor.
- **Date**: The date the payment is due or made.

The data has been pre-processed to provide weekly and monthly summaries, removing any smaller-scale variations while retaining important trends for forecasting.

3.4 Special Payment Arrangements

GS Interfer has several large clients with unique payment arrangements that deviate from the standard payment terms. For example:

- **Client A**: Pays one month later, specifically on the 10th of the following month.
- **Client B**: Pays three months later, also on the 10th of the month.

For these clients, the transaction dates have been adjusted accordingly to reflect their delayed payments. This ensures that the forecasting model captures the actual cash flow dynamics for these clients.

3.5 Data Quality

The dataset does not contain any missing values in either the debtor or creditor transactions. GS Interfer was formed following the merger of Gebro Sales B.V. and Interfer B.V. in 2022. Due to the merger, transaction histories for some customers are missing in the data.

3.6 Cash Flow Calculation

The main target for forecasting in this study is the cash flow, defined as the difference between debtor and creditor amounts for each period. Additionally, cash flow will be forecasted both as a combined metric and separately for debtors and creditors. The goal is to compare the effectiveness of direct cash flow forecasting versus individual debtor and creditor forecasts, combined to derive the cash flow.

3.7 Data Visualizations

To better understand and communicate the patterns in the data, several summary tables and graphs will be used. Tables 3.1, 3.2 and 3.3 summarize the data for cashflow, creditors and debtors respectively. Figure 3.1 to figure 3.6 Show the graphs of the cashflow, the creditor transaction and the debtor transaction on monthly and weekly basis. In the weekly cashflow and debtor transactions the payment arrangements are clearly visible.

	Cashflow Weekly	**Cashflow Monthly**
count	521	120
mean	57610	248003
std	573292	446735
min	-883609	-1055645
25%	-277635	11530
50%	-151833	289988
75%	24889	512649
max	2296156	1686737

Table 3.1: Summary Statistics for Cashflow

	Creditor Weekly	**Creditor Monthly**
count	521	120
mean	-301168	-1310426
std	156339	404904
min	-1000016	-2486140
25%	-372934	-1553428
50%	-272717	-1271818
75%	-192464	-1053151
max	215385	-458883

Table 3.2: Summary Statistics for Creditors

	Debtor Weekly	**Debtor Monthly**
count	521	120
mean	358778	1558428
std	528938	355679
min	-12858	499865
25%	51418	1351157
50%	105218	1537950
75%	182319	1738941
max	2445695	2610594

Table 3.3: Summary Statistics for Debtors

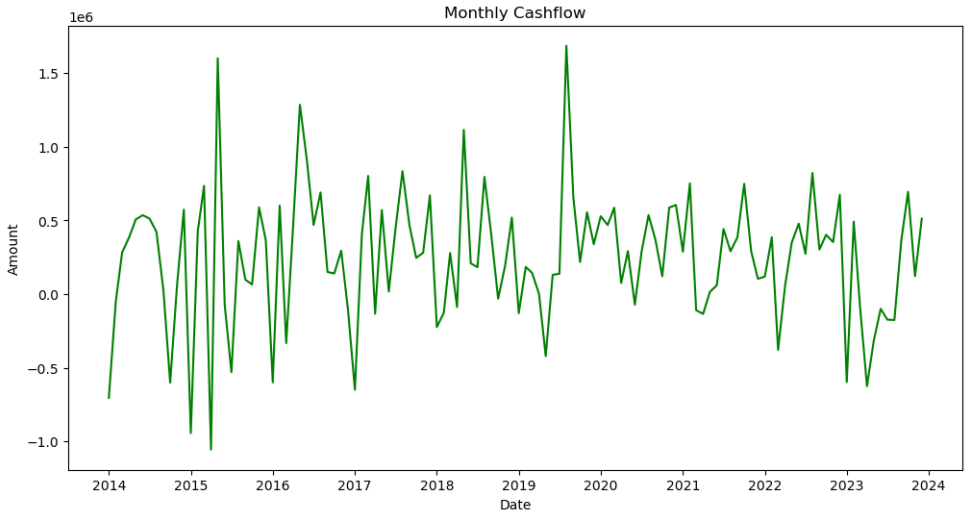


Figure 3.1: Cashflow monthly

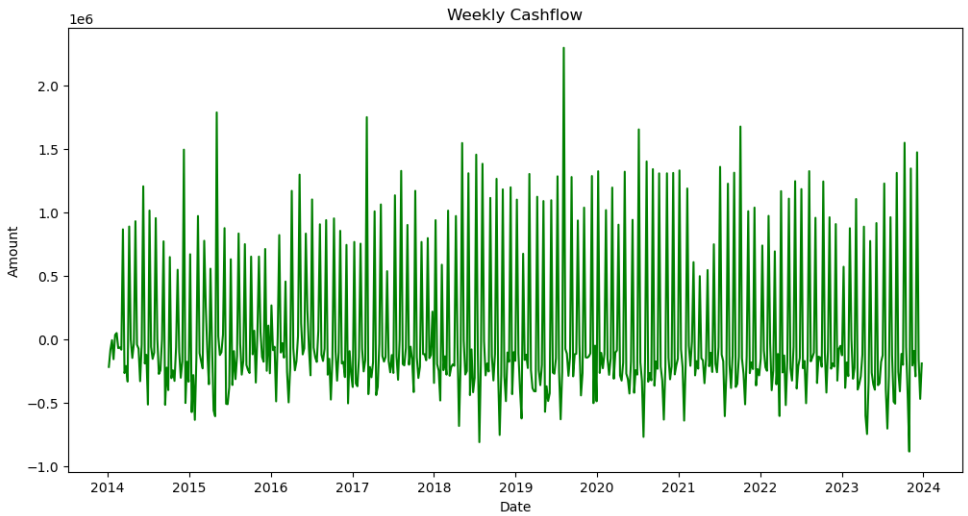


Figure 3.2: Cashflow weekly

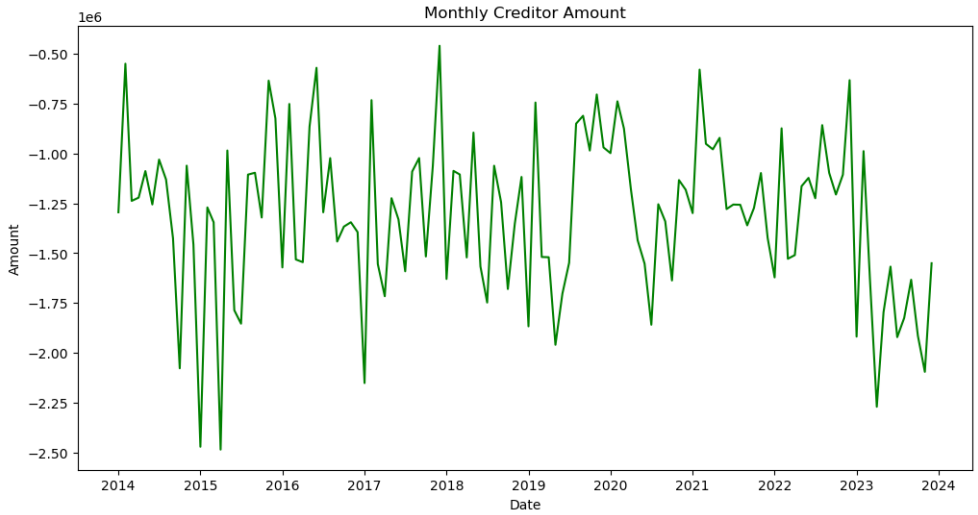


Figure 3.3: Creditor monthly

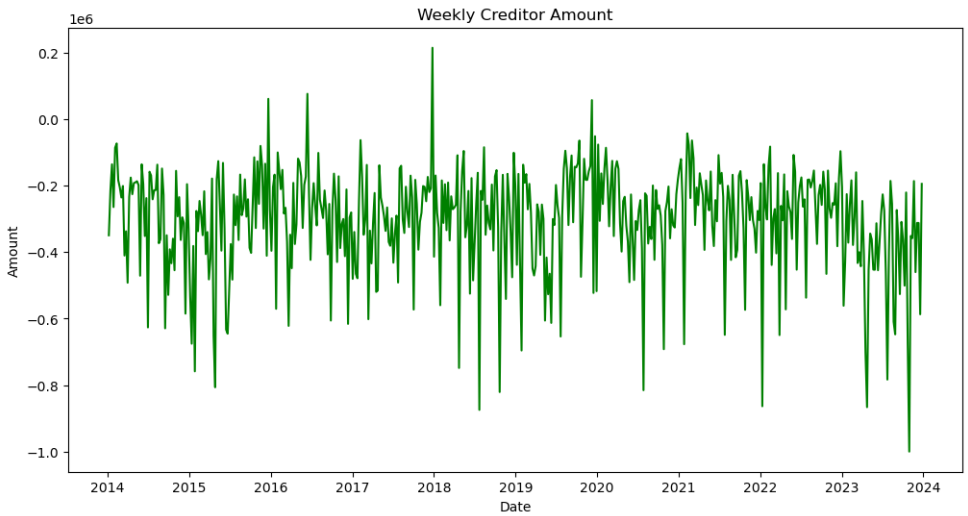


Figure 3.4: Creditor weekly

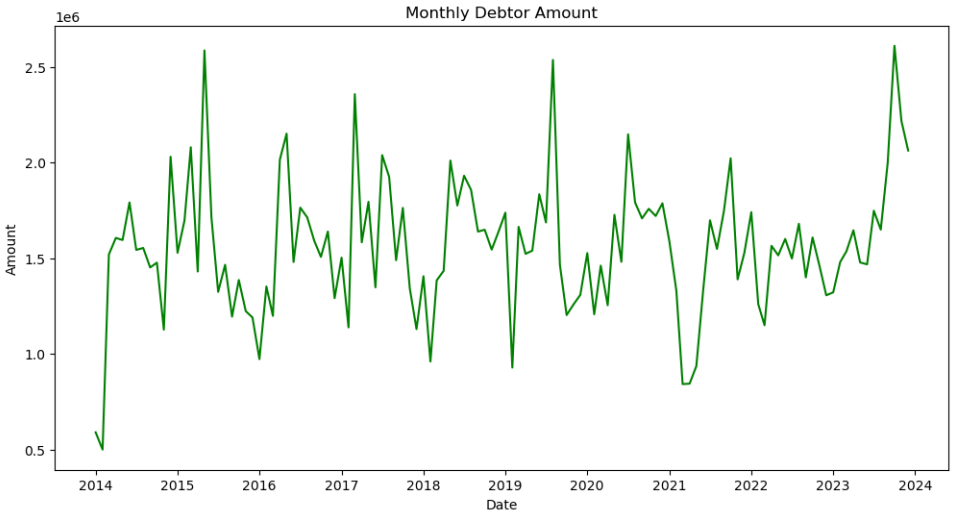


Figure 3.5: Debtor monthly

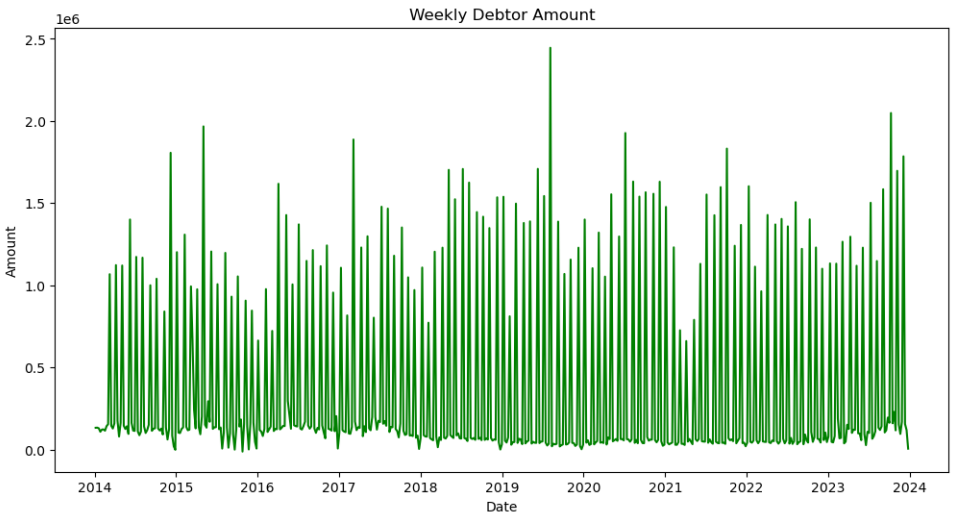


Figure 3.6: Debtor Weekly

Chapter 4

Models

4.1 Exponential Smoothing

This section is dedicated to Forecasting models that use Exponential Smoothing otherwise known as ETS models. These models use weighted averages of previous observations to predict future values. The weight of a previous observation decreases the further back in time the observation was. In other words, more recent observations have higher weights associated with them.

The next subsections cover the framework of exponential smoothing models. First, the methodology is explained. This part covers the different models that can be made with exponential smoothing. The second part is about parameter tuning. It covers how the parameters of the models are optimized and the restrictions they have.

4.1.1 Methodology

ETS models decompose time series into three components: Error, Trend, and Seasonality. Understanding and modeling these components allow analysts to make accurate predictions and gain insight into the underlying patterns in the data. There are several ways to model the components, resulting in different models. Each model comprises of a measurement equation, which describes the observed data, along with some state equations that outline how the unobserved components or states such as level, trend, and seasonality evolve. As a result, these models are commonly referred to as state space models.

To differentiate between the different models we use state space notation ETS(.,.,.) for (Error, Trend, Seasonal). The possibilities for each state is Error $= \{A, M\}$, Trend $= \{N, A, A_d\}$, and Seasonal $= \{N, A, M\}$. An explanation for each is given below.

The Error component represents the random fluctuations or noise in the time series data that cannot be explained by the trend or seasonality. In ETS modeling there are two states for the Error component: Additive (A) and Multiplicative (M). The Main difference between additive and multiplicative errors lies in how they interact with the trend and seasonality components.

Additive errors assume a constant variance independent of the level of the time series, while multiplicative errors vary proportionally with the level of the time series. The choice between additive and multiplicative errors thus depends on the characteristics of the data.

The trend component captures the systematic pattern of the time series. It shows the long-term increase or decrease of the data over time. With exponential smoothing, the trend component allows for adaptive adjustments to changes in the trend's direction or magnitude. In the state space models, there are three options for the trend component: No trend (N), additive trend (A), or damped Additive trend (A_d).

With no trend, the model does not include the trend component. This means that the time series is assumed to have no systematic increase or decrease over time, but is considered to fluctuate around a constant level, exhibiting only random fluctuations and possibly seasonal patterns. With additive trend, the model does contain a trend component and therefore assumes the time series to have a systematic pattern. Forecasts generated with additive trend have a constant trend indefinitely into the future. This can cause problems, especially for longer forecast horizons. Gardner & McKenzie (1985) introduced a parameter that dampens the trend by introducing an extra variable that controls the rate at which the trend approaches zero over time. Therefore, the importance of the trend component diminishes as the forecast horizon extends further into the future. In the state space model, this is damped additive trend.

The seasonality component accounts for the periodic patterns in the time series data that occur at regular intervals. seasonality can be a result of factors such as the season of the year, holidays, or business cycles. In the state space models, there are three options for the seasonality component: No seasonality (N), additive seasonality (A), or multiplicative seasonality (M). In state space models with no seasonality, the seasonal component is not included in the model. This means that the time series is assumed to have no recurring patterns at regular intervals. Instead, any variations in the data are contributed solely to random fluctuations or a possible trend component. With additive and multiplicative seasonality, there is a seasonal component included in the model. The main difference is that additive seasonality assumes constant seasonal fluctuations added to the level, whereas multiplicative seasonality assumes seasonal fluctuations that vary proportionally with the level of the time series.

4.1.2 Parameter Tuning

As stated in the previous section, there are several models that can be made with ETS. These models can use a form of error, trend and seasonality. There are two kinds of parameters that have to be tuned. The first parameters that have to be tuned are the initial state variables level, trend and seasonality. These are the evolving components of the model that are updated with each new observation. The second set of parameters are the smoothing parameters. These parameters determine the balance between actual observations and the previous state variables. These parameters are noted as $\alpha \in [0, 1]$, $\beta \in [0, 1]$, and $\gamma \in [0, 1]$.

α is the level smoothing parameter. It controls the weight given to the most recent observation relative to the smoothed level. With values close to one more weight is given to the most recent data point, allowing the model to react quickly to changes in the data. This means the model will be more responsive to short-term fluctuations. Values close to zero weight is given to the historical smoothed level, meaning the model is less responsive to recent changes, leading to smoother forecasts that may not capture recent shifts in the data as quickly.

β is the trend smoothing parameter. This controls how much weight is given to the most recent trend estimate in updating the trend component of the model. With values close to 1, the model adapts quickly to changes in the trend, making it more sensitive to shifts in the trend over time. with values close to zero, the model reacts more slowly to changes in the trend and gives more weight to the historical trend, making it more stable but less responsive to recent changes.

γ is the seasonal smoothing parameter. It controls how much weight is given to the most recent seasonal observation when updating the seasonal component. values close to one give more weight to the most recent seasonal effect, making the model more responsive to recent seasonal variations. When values are close to zero the model is less sensitive to recent changes in seasonality and gives more weight to the historical seasonal pattern, leading to more stable but less responsive seasonal adjustments.

4.2 SARIMA

SARIMA models or Seasonal Auto Regressive Integrated Moving Average uses a different approach to forecasting than Exponential Smoothing. As stated in the previous section, Exponential smoothing combines a trend and seasonality component to forecast future values. SARIMA models use autocorrelations within the data to make predictions. It is important with SARIMA that the underlying data is stationary. Before the methodology is discussed, the concept of Stationarity and Differencing is explained.

4.2.1 Stationarity and Differencing

Stationarity is an important concept in time series analysis. Stationary time series are time series whose statistical properties do not depend on time. So, the behavior of the time series does not change with time. This means that series with trends, or seasonality are not stationary, as they influence the data differently at different times. A series with cyclic behavior, but no trend and seasonality, is stationary. This is due to the uncertainty that arises because the cycles in the series do not have a fixed length. Therefore, until we observe the series, it's impossible to determine the precise locations of the peaks and troughs within these cycles. An example of a stationary series is a white noise series.

A common technique to make a non-stationary time series stationary is called differencing. It involves computing the difference between two consecutive observations or between an observation and the previous observation from the same season. Differencing helps to stabilize the mean of a time series by removing or reducing changes in the level of the series. It therefore reduces the trend or seasonality component.

Taking the difference between two consecutive observations is sometimes called "first differences" as the difference is taken with lag one. This means that the new series looks like this:

$$y_t^{'} = y_t - y_{t-1} \tag{4.1}$$

Sometimes taking only the difference with lag one is not enough to make the series stationary. It is then necessary to difference the data again. This is called "second-order differencing". This means that the change in changes is modeled. The new series looks like:

$$y_t^{''} = y_t^{'} - y_{t-1}^{'} = y_t - 2y_{t-1} + y_{t-2} \tag{4.2}$$

Seasonal differencing involves subtracting observations from earlier seasonal periods. So this is written as:

$$y_t^{'} = y_t - y_{t-m} \tag{4.3}$$

, where m is the number of seasons in a year. These are also called "lag-m differences". For example: a time series with monthly data has 12 data points

within a year, so $m = 12$. Seasonal differencing, in this case, means subtracting last year's observation of the same month from this year's observation. It is also possible for series to need seasonal differencing and first differencing. let $y_t^{'}$ be the seasonal differenced data, Then the twice-differenced series looks as follows:

$$y_t^{''} = y_t^{'} - y_{t-1}^{'} = (y_t - y_{t-m}) - (y_{t-1} - y_{t-m-1}) = y_t - y_{t-1} - y_{t-m} + y_{t-m-1} \quad (4.4)$$

The order in which the differencing is applied makes no difference. It is however recommended to first use seasonal differencing because the resulting series might be stationary and there is no need for further differencing. Taking more difference than necessary can cause false autoregressions and should therefore be avoided.

There is an amount of subjectivity in choosing which differencing to use, as the modeling process requires choices. A more objective way to decide when to use differencing is to do a test. In this thesis two statistical methods are used to determine whether a time series is stationary or non-stationary.

The first statistical test is KPSS (Kwiatkowski-Phillips-Schmidt-Shin) test (Kwiatkowski et al., 1992). The null hypothesis of the KPSS test is that the series is stationary. The alternative hypothesis is that the series is non-stationary. A p-value that is less than the desired significance level means that the null hypothesis is rejected, resulting in the conclusion that the series is non-stationary, and thus needs differencing. If the p-value is greater than the desired significance level, the test fails to reject the null hypothesis. This means that there is insufficient evidence to reject the null hypothesis. The KPSS test returns a p-value between 0.01 and 0.1, however, the actual p-value could be higher or lower.

The second statistical test used in this thesis is the ADF (Augmented Dickey-Fuller) test. The null hypothesis is that the time series has a unit root, which means that it is non-stationary. A p-value greater than a significance threshold would suggest that you cannot reject the null hypothesis, meaning the series is likely non-stationary.

Both test are used in this thesis. This thesis assumes that a time series is stationary when the KPSS test has a p-value of higher than 0.05 and the ADF test has a p-value lower than 0.05

To more easily use differencing in formulas, the backward shift operator B is introduced.

$$By_t = y_{t-1} \quad (4.5)$$

When two backward shifts are used, the data is shifted back two periods

$$B(By_t) = B^2 y_t = y_{t-2} \quad (4.6)$$

This means that the data of the previous season is denoted as

$$B^m y_t = y_{t-m} \quad (4.7)$$

A first difference is denoted as

$$y_t^{'} = y_t - y_{t-1} = y_t - By_t = (1 - B)y_t \quad (4.8)$$

Similarly, the second order difference is written as

$$y_t^{''} = y_t - 2y_{t-1} + y_{t-2} = y_t - 2By_t + B^2y_t = (1-B)^2y_t \tag{4.9}$$

In general, a difference of order d can be written as

$$(1-B)^d y_t \tag{4.10}$$

A seasonal difference followed by a first difference is denoted as

$$(1-B)(1-B^m)y_t = 1 - B - B^m + B^{m+1} = y_t - y_{t-1} - y_{t-m} + y_{t-m-1} \tag{4.11}$$

4.2.2 Methodology

SARIMA combines an auto-regressive model with a moving average model. In the next subsections, it is explained what these models entail en how SARIMA combines them.

Auto Regressive Model

Auto-regressive models predict future values of the desired variable using a weighted sum of its past values, plus a random error term. An autoregressive model of order p can be written as

$$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + ... + \phi_p y_{t-p} + \epsilon_t \tag{4.12}$$

, where y_t is the desired variable at time t, c is the intercept, $\phi_1, ..., \phi_p$ are autoregressive coefficients, representing the weights of the past observations, and ϵ_t is the error term at time t which is assumed to have zero mean and a constant variance. This is referred to as an $AR(p)$ model. The p in the $AR(p)$ model determines the number of lagged variables in the model.

Moving Average Model

Moving Average models are similar to Auto-Regressive models. The difference is that where Auto-Regressive models use past values of the desired variable itself, whereas Moving Average models use past forecast errors. A model of order q can be written as

$$y_t = c + \epsilon_t + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + ... + \theta_q \epsilon_{t-q}$$

, where y_t is the desired variable at time t, c is the intercept, $\theta_1, ..., \theta_q$ are the moving average coefficients, representing the weights assigned to the past forecast errors, and ϵ_t is the forecast error at time t which is assumed to have zero mean and a constant variance. This is referred to as an $MA(q)$ model. The q in the $MA(q)$ model determines the number of past forecast errors used in the model. Notice that this is not a regression in the typical sense as ϵ_t is not observed

ARIMA

The result of combining differencing with auto-regressive models and moving average models is a non-seasonal ARIMA model. This model can be written as:

$$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + ... + \phi_p y_{t-p} + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + ... + \theta_q \epsilon_{t-q} + \epsilon_t \quad (4.13)$$

, where y_t is the desired variable at time t, c is the intercept, $\phi_1, ..., \phi_p$ are the autoregressive coefficients, $\theta_1, ..., \theta_q$ are the moving average coefficients, and ϵ_t is the forecast error at time t which is assumed to have zero mean and a constant variance. This is referred to as an $ARIMA(p, d, q)$ model. Where p determines the number of lagged variables, q determines the number of past forecast errors, and d is the amount of times first differencing is used.

To work with this formula, it is convenient to convert it to backshift notation.

$$(1 - \phi_1 B - ... - \phi_p B^p)(1 - B)y_t = (1 - \theta_1 B - ... - \theta_q B^q) \quad (4.14)$$

$$(1 - \phi_1 B - ... - \phi_p B^p)(1 - B)y_t = (1 - \theta_1 B - ... - \theta_q B^q)$$

SARIMA

It is possible to extent the ARIMA model with seasonal components, this is often referred to as SARIMA. SARIMA incorporates seasonal terms that mirror the structure of non-seasonal components in ARIMA but apply to seasonal lags. The length of this seasonal lag is denoted by s. For monthly data $s = 12$ and for weekly data $s = 52$. The SARIMA model is denoted as $SARIMA(p, d, q)(P, D, Q)s$, where p, d and q represent the non seasonal components that are also present in the ARIMA model. P, D and Q represent the order of the seasonal auto-regresson, seasonal differencing and seasonal moving average.

Seasonal auto-regression captures correlations between observations separated by the seasonal period. Seasonal moving average captures patterns in the residuals from previous seasonal periods. Seasonal differencing is similar to differencing but applied to remove seasonal trends.

4.2.3 Parameter Tuning

There are several parameters that can be tuned for SARIMA models. First the KPSS and the ADF test are used to see whether the data is stationary. If the data needs seasonal differencing, $D = 1$. If the data need differencing, $d = 1$. After that the values of p, q, P, and Q can be tuned. To get feasible results, ACF and PACF can be used, however this does not guaranty that these are the optimal values for the parameters. Therefore a grid search is used.

4.3 LSTM Neural Network

In this chapter, the cash flow will be forecasted with a long short-term memory network or LSTM. This is a form of artificial neural network (ANN). To understand the theory behind the LSTM, it is essential to first explain the methodology of the ANN.

Figure 4.1 visualizes a possible configuration of a neural network. A neural network contains neurons and layers. There are three types of layers. The first layer on the left is the input layer. This is where the data is inputted in the model. The second layer is the hidden layer. It is possible to have multiple hidden layers, however, for simplicity, there is only one hidden layer used in the example. The last layer is called the output layer. This is where the forecast of the cash flow is produced.

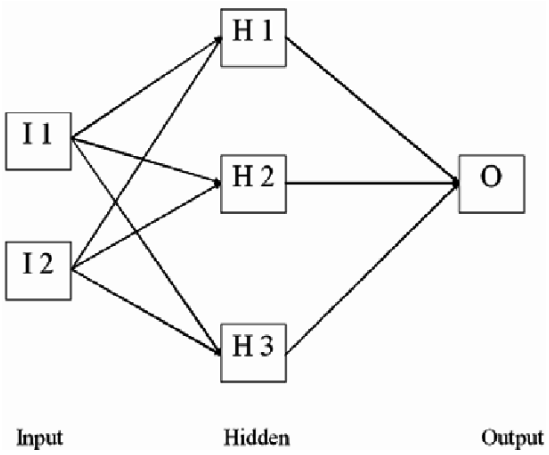


Figure 4.1: Simple neural network

As shown by the arrows in Figure 4.1, the data in a neural network flows from left to right. Each neuron is connected to all the neurons in the next layer. The connection between two neurons is given by a weight w. The weight between neurons determines the importance of that connection. To get the value of a neuron other than the input neurons, the weighted sum is taken of all input values and a bias term is added. The resulting weighted sum for each node is given by equation 4.1. In this equation, z is the weighted sum, w_i is the weight associated with input i, x_i is the value of input i, and b is the bias term for this neuron. The bias term is added to make adjustments to the model so it can fit the data better. After that, an activation function g is used to capture non-linearity (Efron and Hastie, 2021).

$$z = \sum_{i}^{n} (w_{i=1} \times x_i) + b \tag{4.15}$$

In equations 4.2 and 4.3 below, the vector notation is given for the weighted sum before and after the activation function. In equation 4.2, W^{k-1} represents the matrix of weights going from layer $l-1$ to layer l. In the example of figure 4.1, the matrix W^1 would be a 2×3 matrix, since the are 2 neurons in the

input layer and 3 neurons in the hidden layer. In equation 4.3, the non-linear activation function g^l is used. This function is applied element-wise, which means the function is used on each neuron individually. The result of equation 4.3 is called the activation vector denoted by $a^{(l)}$.

$$z^l = W^{l-1}a^{l-1} + b^{l-1} \tag{4.16}$$

$$a^l = g^l(z^l) \tag{4.17}$$

To get a forecast for cash flow, the data has to go through the neural network. This happens layer by layer. For each layer, equations 4.2 and 4.3 are used to calculate the values of the neurons in that layer. This happens until the output layer is reached. This process is called forward propagation.

The neural network learns by adjusting the weights and biases. It does this by minimizing a loss function denoted by $L[y, f(x)]$. Where L denotes the loss function, y denotes the true value, and $f(x)$ denotes the output of the model. There are several different loss functions. For this thesis, the mean squared error is used. The mean squared error is denoted in equation 4.4. A loss function aims to calculate a score on how wrong the output of the model is compared to the true value. For mean squared error, larger errors are punished more heavily. There are n observations in the training data. In equation 4.5 the objective function of a neural network is given.

$$L[y_i, f(x_i; W)] = \frac{1}{n}\sum_{i=1}^{n}(y_i - f(x_i; W))^2 \tag{4.18}$$

$$\min_W \{\frac{1}{n}\sum_{i=1}^{n} L[y_i, f(x_i; W)]\} \tag{4.19}$$

To train the model, backpropagation, and gradient descent are used. First, the weights and biases must be initialized. This is done by randomly assigning each weight and bias a normally distributed value with zero mean and a small variation. Secondly, forward propagation is done to get a value for each neuron. The next step is to compute the error term δ^l. Now that the output and the loss are computed, the weights and biases can be updated such that the loss is reduced. This is where backpropagation is used. Backpropagation relies heavily on the chain rule. For any function that depends on intermediate variables, we can compute derivatives layer-by-layer. As the name suggests, backpropagation starts at the output layer and works backward to the first hidden layer. The derivative of the output layer is given by equation 4.6.

$$\frac{\partial L}{\partial z^{[L]}} = \frac{\partial L}{\partial a^{[L]}} \cdot g^{(l)\prime}(z^{[L]}) \tag{4.20}$$

Where L denotes the loss function, $z^{[L]}$ denotes the vector of weighted sums in the output layer and $g^{(l)\prime}(z^{[L]})$ denotes the derivative of the activation function at layer L. For each of the hidden layers, the error term δ^l is given by equation 4.7.

$$\delta^{[l]} = \left(W^{[l+1]}\delta^{[l+1]}\right) \odot g^{(l)\prime}(z^{[l]}) \tag{4.21}$$

Where $\odot$ denotes element-wise multiplication. After all $\delta^{[l]}$ are calculated for each layer the derivatives for the weights and biases for each layer l can be calculated as denoted in equation 4.8 and 4.9 respectively.

$$\frac{\partial L}{\partial W^{[l]}} = \delta^{[l]} \cdot (a^{[l-1]})^T \tag{4.22}$$

$$\frac{\partial L}{\partial b^{[l]}} = \delta^{[l]} \tag{4.23}$$

Finally, the gradient descent will be applied to update the weights and biases. The updates are shown in equations 4.10 and 4.11.

$$W^{[l]} \leftarrow W^{[l]} - \lambda \frac{\partial L}{\partial W^{[l]}} \tag{4.24}$$

$$b^{[l]} \leftarrow b^{[l]} - \lambda \frac{\partial L}{\partial b^{[l]}} \tag{4.25}$$

Where $\lambda \in (0, 1]$ is the learning rate. The learning rate determines the speed at which the values are updated. This process of backpropagation is repeated until a certain stopping criterion is met.

4.3.1 Methodology

So far the basic concept of an artificial neural network is explained. However, in this thesis, LSTM networks are used. LSTM is a form of a neural network more suited for time series data. The next part will explain why LSTM is more suited for the data used in this thesis.

ANNs are feedforward networks, which means that they only process information in one direction and treat each input as independent of the other. However, in time series data, each point is dependent on previous points. Because of this ANNs have no memory or awareness of previous inputs, which limits the ability to recognizance trends and long-term dependencies.

Recurrent Neural Networks (RNN) contain internal memory that allows them to retain information about previous inputs. This makes them suitable for handling time-dependent data, where past information can inform future predictions. In Figure 4.2 a RNN is visualised. The theory behind an RNN is similar to that of an ANN, however, a RNN has a feedback loop. This feedback loop makes it possible for the network to have a memory. On the right side of the figure, the network is unfolded. Here it is shown that the network can be seen as multiple copies of a structure where information can be passed between them.

Similar to an ANN, backpropagation en gradient descent are used to train the network. This can however cause a problem called vanishing or exploding gradients. Vanishing gradients are caused by backpropagating gradients in a long sequence, gradients can shrink to a value close to zero, causing the network to "forget" earlier parts of the sequence. Similarly exploding gradients happen when gradients become excessively large during backpropagation, they cause the weights to grow uncontrollably, leading to instability in the model.

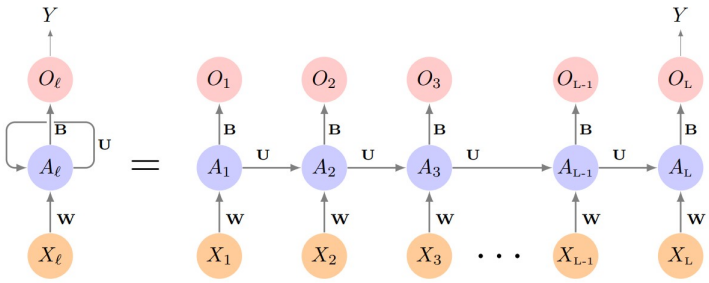


Figure 4.2: Recurrent neural network

The reason this happens in RNNs is because the same weights are repeatedly applied across each time step in a sequence, effectively creating a very deep network where the number of layers corresponds to the length of the sequence. If the gradients of these repeated layers are even slightly greater than one, they can grow exponentially with the length of the sequence.

To mitigate this problem, a long short-term network is proposed by Hochreiter and Schmidhuber (1997). LSTMs introduce a mechanism called gated memory cells that allow the network to decide which information to retain, update, or discard over time. This mechanism not only dissolves the vanishing or exploding gradient problem but also allows the network to remember relevant long-term patterns while discarding irrelevant information.

In Figure 4.3 the network of a LSTM is visualized. The network contains three input components. The first component is the cell state c_t. This is where the long-term memory is saved. The second component is the hidden state h_t. This represents the short-term memory and is updated at each time step. The third component is the input at time t x_t. Three gates control how the information flows in the network. The first gate is the forget gate f_t. This gate decides what part of the previous cell state should be forgotten. The second gate is the input gate i_t. This controls what new information should be added to the cell state. The third gate is the output gate o_t. This gate controls what part of the cell state should be outputted to the hidden state. All gates will be explained further.

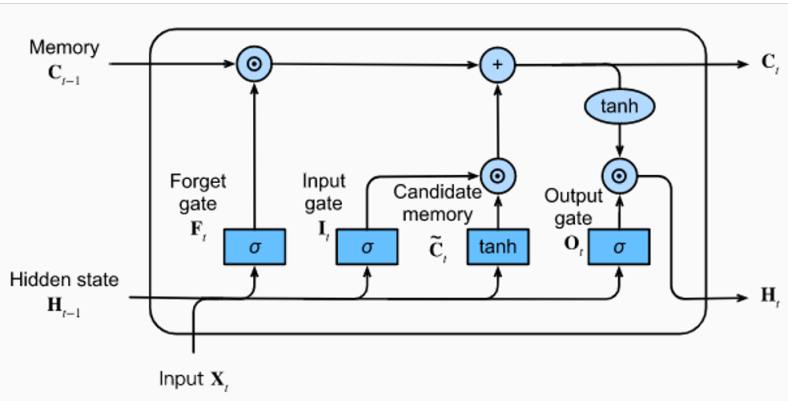


Figure 4.3: LSTM neural network

Before the gates are explained further, there are 2 activation functions used in the network that are important to understand. Therefore, these functions are explained first. The two functions are the sigmoid function and the hyperbolic tangent function. In Figure 4.3, these functions are symbolized by σ and tanh respectively. The sigmoid function turns a value $x \in \mathbb{R}$ to a value $y \in [0, 1]$ and the hyperbolic tangent function turns a value $x \in \mathbb{R}$ to a value $y \in [-1, 1]$. The functions are shown in Figure 4.4 and denoted in equations 4.12 and 4.13 respectively.

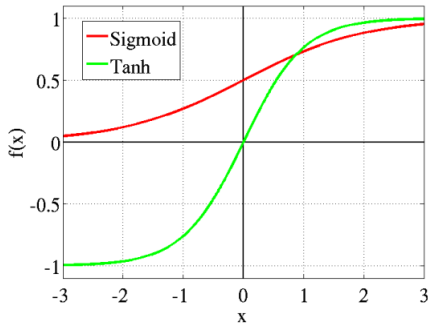


Figure 4.4: sigmoid and tanh function

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{4.26}$$

$$tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{4.27}$$

Now that the functions are clear, the gates can be explained. First the forget gate. The forget gate has two inputs x_t and h_{t-1}. These inputs are multiplied by their weights and summed, and the bias is added to get a result. This result is then used as input for the sigmoid function, which converts it into a value between zero and one. This output is then multiplied by c_{t-1}. Because the output of the sigmoid function is a number between zero and one, the forget gate can be seen as the gate where a percentage of the long-term memory is forgotten. The forget gate is given by equation 4.14. Where W_f is the weight matrix of the forget gate, h_{t-1} is the hidden state at time $t-1$ and x_t is the input at time t.

$$f(t) = \sigma(W_f \times h_{t-1} + W_f * x_t) \tag{4.28}$$

The second gate is the input gate. This gate has two inputs x_t and h_{t-1}. The input gate contains two blocks that together determine the new information that is added to the long-term memory. The first block determines the percentage of the potential memory to remember. Similar to the forget state, the inputs are multiplied by weights, and summed, and the bias is added to get a result. This result is then used as input for the sigmoid function, which converts it into a value between zero and one. This is similar to the forget gate, however, the weights and biases differ, resulting in a different outcome.

This is shown in equation 4.15 The second Block determines the potential long-term memory. The inputs are again multiplied by weights and summed, and the bias is added to get a result. The weights and bias are different for this block which results in a different outcome. This result is then used as input for the hyperbolic tangent function resulting in an output in the range of $[-1, 1]$. The hyperbolic tangent function is used because it makes negative values possible. This is necessary when dealing with a negative trend. This is shown in equation 4.16. The output of the input gate is the multiplication of the output of the two blocks. To get the new long-term memory the output of the forget gate and input gate are added. This is shown in equation 4.17.

$$i_t = \sigma(W_i \times h_{t-1} + W_i \times x_t + b_i) \tag{4.29}$$
$$\tilde{c}_t = \tanh(W_c \times h_{t-1} + W_c \times x_t + b_c) \tag{4.30}$$
$$c_t = f_t \times c_{t-1} + i_t \cdot \tilde{c}_t \tag{4.31}$$

Lastly, the output gate is explained. The output gate has the same input as the other gates x_t and h_{t-1}. In this gate, the same procedure is applied. The equation is given by 4.18. This output is multiplied with the hyperbolic tangent function of c_t. This is given by equation 4.19.

$$o_t = \sigma(W_o \times h_{t-1} + W_o \times x_t + b_i) \tag{4.32}$$
$$h_t = o_t \times \tanh(c_t) \tag{4.33}$$

Algorithm 1 Long short-term memory network

Ensure: : Training data set with independent variables p

y; A vector with true values
λ; Learning rate
E; Number of iterations
Give weights W_f, W_i and W_o random starting values such that $W \sim \mathcal{N}(0, 0.1)$
for e = 1 to E **do**
    Forward propagation
    Backpropagation w.r.t. all weights
    Apply gradient descent for all gates;
    $W_f \leftarrow W_f + \lambda \delta W_f$
    $W_i \leftarrow W_i + \lambda \delta W_i$
    $W_o \leftarrow W_o + \lambda \delta W_o$
    if Stopping criteria is met **then**
        Break the for loop
    end if
end for

4.3.2 Parameter Tuning

To develop a log short-term network, several important parameters have to be tuned. Optimizing these parameters can improve predictive performance. In this section, the parameters that need tuning are explained.

The first important parameter of the LSTM that needs to be tuned is the number of hidden layers. An increase in hidden layers also means an increase in the complexity of the network. This can help the model find more difficult historical patterns. However, this comes with a cost as it can cause overfitting and has a high computational demand. The second parameter that has to be tuned is the number of neurons in the hidden layers. More neurons in each layer increase the complexity of the model such that it can learn complex relationships, however, too many neurons can lead to overfitting. Stathakis (2009) has investigated how many hidden layers and nodes are optimal for a neural network. First, the number of neurons should be between the size of the input and the size of the output. The range for the number of hidden layers is smaller. For any non-linear complex problem, one or two hidden layers are sufficient.

The third parameter is the learning rate. This parameter is necessary for gradient descent and determines the speed at which the model converges to an optimal solution. Small learning rates provide stable convergence, however, if the learning rate is too small the local minimum might not be reached. Larger learning rates accelerate learning but might lead to instability. In this thesis, the Adam optimizer is used during the learning process. The Adam optimizer or Adaptive Moment Estimation, is well suited for training neural networks as it computes individual adaptive learning rates for different parameters.

The fourth parameter that has to be tuned is the amount of epochs. An epoch refers to a single pass through the entire training dataset by the model during the learning process. During each epoch, the LSTM processes all training data once and updates its weights to minimize prediction errors. Multiple epochs are typically required to allow the model to learn effectively from the data, as each epoch refines the model's understanding of underlying patterns. By using a higher number of epochs, an LSTM can improve its accuracy, but too many epochs can lead to overfitting. To prevent this an early stopping criteria is used. If the error is not reduced for several epochs then the stopping criteria are met and the model will stop training.

Another parameter that is used in LSTM is the amount of lag variables. Lag variables refer to the amount of previous data points as input. For this thesis, the amount of lag variables used for LSTM is ten.

For neural networks, it is possible to divide the data in batches. This means that the network is not trained one data point at a time, but that we calculate the gradient descent over several data points. This reduces the amount of memory used thus decreasing the computation time. With small batches, each update to the model parameters is based on a limited sample of the data. This introduces more noise in the gradient descent steps, which can help the model generalize better. Small batches require more iterations to process the same amount of data, potentially making training slower. Larger batches produce

more stable gradient updates, which can make the model converge faster per epoch. Keskar et al. (2017) found that large-batch training can negatively impact generalization in deep learning models, leading them to converge to sharp minima that overfit the training data but fail to generalize well to new data.

It is important to scale the data for the last network. As a consequence of using the sigmoid and the hyperbolic tangent function, the data needs to be normalized. Therefore the data will be normalized with zero mean and a standard deviation of one. To prevent the look-ahead bias, the mean and deviation will only be calculated over the training data and not the test data. If the test data was used in the normalization, then the network would have information about the future. In addition to this, every batch is also normalized. Batch normalization, researched by Ioffe and Szegedy (2015), reduces internal covariate shifts and accelerates convergence. The method normalizes the activations of each layer using the mean and variance from the batch, enabling the use of higher learning rates, and in some cases eliminating the need for Dropout.

Chapter 5

Results

5.1 Evaluation Criteria

To evaluate the forecasts made in this thesis there need to be evaluation criteria. All forecasts are evaluated on three criteria. These are the Root Mean Squared Error (RMSE), the Mean Absolute Error (MAE), and the Mean Absolute Percentage Error. These criteria are given by equations 5.1-5.3. RMSE punishes larger errors more heavily, while MAE punishes proportional to the error. MAPE is a percentage error which means that it punishes errors concerning the true value. Percentage errors have the disadvantage of being undefined if the true value is zero and having extreme values if true values are close to zero. In some of the tables in the section Numerical Results, this is the case. They also have the disadvantage that they put a heavier penalty on negative errors than on positive errors.

$$\text{RMSE} = \sqrt{\frac{1}{T}\sum_{t=1}^{T}(y_t - \hat{y}_t)^2} \tag{5.1}$$

$$\text{MAE} = \frac{1}{T}\sum_{t=1}^{T}|y_t - \hat{y}_t| \tag{5.2}$$

$$\text{MAPE} = \frac{1}{T}\sum_{t=1}^{T}|\frac{y_t - \hat{y}_t}{y_t}| \tag{5.3}$$

5.2 Numerical results

In this section, the results are shown for all forecasts made. In each section, results are shown on a weekly and monthly basis for each method. For cases where parameter tuning was necessary, there are also tables to show the optimal value.

5.2.1 Exponential Smoothing

In this section, the results for exponential smoothing are shown. Parameter tuning is optimized by the Python function used. Therefore the only tables that are shown are for the evaluation criteria.

Results per time series

For monthly cashflow, the Season and trend method is best in terms of RMSE and MAE. For the weekly data, the Season and trend method is best in terms of RMSE and MAE. For both monthly and weekly data the models that contain seasonality perform better. For the monthly creditor transactions, the damped trend model performs best in all criteria. For the weekly creditor transactions, the model with a seasonality component but no trend component is best for all criteria. For Monthly debtor transactions, the season without trend model is best for RMSE, but the damped trend method is best for MAE and MAPE. For Weekly debtor transactions, the season and no trend model is best for RMSE and MAE.

Cashflow Monthly	**RMSE**	**MAE**	**MAPE**
ETS - Simple	517092	441094	189
ETS - Trend	545017	456866	206
ETS - Trend Damped	518546	441898	190
ETS - Season no trend	465784	368964	158
ETS - Season and trend	455463	363908	154
ETS - Season and damped trend	467206	370679	160

Table 5.1: Forecasting model performance of ETS models

Cashflow Weekly	**RMSE**	**MAE**	**MAPE**
ETS - Simple	630411	529752	139
ETS - Trend	633109	537716	154
ETS - Trend Damped	630194	528531	138
ETS - Season no trend	435740	284253	102
ETS - Season and trend	432398	278063	111
ETS - Season and damped trend	437268	287058	102

Table 5.2: Forecasting model performance of ETS models

Creditor Monthly	**RMSE**	**MAE**	**MAPE**
ETS - Simple	777813	722277	39
ETS - Trend	915774	856712	46
ETS - Trend Damped	629809	584452	32
ETS - Season no trend	636909	590320	33
ETS - Season and trend	767267	730482	41
ETS - Season and damped trend	668165	624268	35

Table 5.3: Forecasting model performance of ETS models

Creditor Weekly	**RMSE**	**MAE**	**MAPE**
ETS - Simple	235121	171129	36
ETS - Trend	257603	192786	40
ETS - Trend Damped	240995	176973	37
ETS - Season no trend	226459	166355	37
ETS - Season and trend	228494	167879	37
ETS - Season and damped trend	226574	166455	37

Table 5.4: Performance of ETS Models (Test Data)

Debtor Monthly	**RMSE**	**MAE**	**MAPE**
ETS - Simple	502188	362477	18
ETS - Trend	527723	388804	19
ETS - Trend Damped	483639	339517	16
ETS - Season no trend	464656	357688	18
ETS - Season and trend	531089	395523	20
ETS - Season and damped trend	475379	364532	18

Table 5.5: Performance of ETS Models (Test Data)

Debtor Weekly	**RMSE**	**MAE**	**MAPE**
ETS - Simple	573853	422770	360
ETS - Trend	607972	362762	191
ETS - Trend Damped	571140	436214	391
ETS - Season no trend	414215	215407	107
ETS - Season and trend	416566	221776	109
ETS - Season and damped trend	416449	220942	112

Table 5.6: Forecasting model performance of ETS models

Conclusion Exponential Smoothing

Tables 5.7 and 5.8 show the best-performing models for each time series based on RMSE and MAE respectively together with the evaluation criteria. To see what the best methods are to forecast the cash flow with ETS based on RMSE and MAE, the evaluations of the cash flow and combined cash flow are compared. This is shown in Tables 5.9 and 5.10. Week cash flow is the forecast made with the Season and trend model. Week combined is a forecast created by combining the best forecasts of Credit week and Debit week. Month cashflow and Month combined are made in a similar matter. According to Tables 5.9 and 5.10, it is best to use the original forecast for weekly and monthly data.

Time Series	**Model**	**RMSE**	**MAE**	**MAPE**
Cashflow Monthly	ETS - Season and trend	455463	363908	154
Cashflow Weekly	ETS - Season and trend	432398	278063	111
Creditor Monthly	ETS - Trend Damped	629809	584452	32
Creditor Weekly	ETS - Season and damped trend	226574	166455	37
Debtor Monthly	ETS - Season no trend	464656	357688	18
Debtor Weekly	ETS - Season no trend	414215	215407	107

Table 5.7: Best Performing Models Based on RMSE

Time Series	**Model**	**RMSE**	**MAE**	**MAPE**
Cashflow Monthly	ETS - Season and trend	455463	363908	154
Cashflow Weekly	ETS - Season and trend	432398	278063	111
Creditor Monthly	ETS - Trend Damped	629809	584452	32
Creditor Weekly	ETS - Season and damped trend	226574	166455	37
Debtor Monthly	ETS – Trend Damped	483639	339517	16
Debtor Weekly	ETS - Season no trend	414215	215407	107

Table 5.8: Best Performing Models Based on MAE

Method	**RMSE**	**MAE**	**MAPE**
Week Combined	433912	285545	106
Week Cashflow	432397	278062	110
Month Combined	564972	486437	207
Month Cashflow	455463	363908	154

Table 5.9: Forecasting model performance of ETS models

Method	RMSE	MAE	MAPE
Week Combined	433912	285545	106
Week Cashflow	432397	278062	110
Month Combined	473765	414489	162
Month Cashflow	455463	363908	154

Table 5.10: Forecasting model performance of ETS models

5.2.2 SARIMA

In this section, the results for AARIMA are shown. Parameter tuning is necessary. To determine if the time series are stationary the ADF test and the KPSS test are used. Table 5.11 shows that all series are stationary. This means that differencing is not necessary. Now the other parameters are tuned with grid search. To be complete d and D are allowed to be one. For each time series, the top 5 best performing SARIMA models based on RMSE and MAE are shown in tables 5.12-5.16 and tables 5.17-5.22 respectively.

(p,d,q)	(P,D,Q,s)	RMSE
(0,0,1)	(1,1,1,52)	210240
(1,0,0)	(0,1,0,52)	230894
(0,1,1)	(0,0,0,52)	236083
(2,1,2)	(0,0,0,52)	237119
(0,1,1)	(1,0,1,52)	238720

Table 5.11: Top 5 SARIMA models based on RMSE for week credit

(p,d,q)	(P,D,Q,s)	RMSE
(2,1,2)	(1,0,1,52)	406033
(2,1,0)	(1,1,0,52)	408165
(2,1,0)	(0,1,1,52)	408763
(0,1,0)	(1,1,0,52)	409173
(1,0,1)	(0,1,1,52)	413447

Table 5.12: Top 5 SARIMA models based on RMSE for week debit

(p,d,q)	(P,D,Q,s)	RMSE
(1,0,2)	(0,1,0,52)	440998
(1,0,2)	(1,0,1,52)	441112
(2,0,1)	(1,1,1,52)	441677
(2,1,1)	(0,1,0,52)	441988
(2,0,2)	(0,0,1,52)	486562

Table 5.13: Top 5 SARIMA models based on RMSE for week cashflow

(p,d,q)	(P,D,Q,s)	RMSE
(0,0,2)	(0,1,0,12)	686409
(2,0,2)	(0,1,1,12)	717383
(1,0,1)	(0,1,1,12)	727650
(1,1,1)	(0,0,1,12)	886904
(0,1,0)	(1,1,1,12)	944404

Table 5.14: Top 5 SARIMA models based on RMSE for month credit

(p,d,q)	(P,D,Q,s)	RMSE
(1,0,2)	(0,1,1,12)	481184
(2,0,2)	(0,1,1,12)	483981
(1,1,1)	(0,1,1,12)	514551
(1,0,2)	(1,1,0,12)	517414
(2,1,1)	(0,0,1,12)	533299

Table 5.15: Top 5 SARIMA models based on RMSE for month debit

(p,d,q)	(P,D,Q,s)	RMSE
(2,0,0)	(1,0,1,12)	419787
(1,1,1)	(1,1,0,12)	430372
(1,1,1)	(1,1,1,12)	456482
(1,0,1)	(1,0,1,12)	460481
(2,0,2)	(0,1,1,12)	480436

Table 5.16: Top 5 SARIMA models based on RMSE for month cashflow

(p,d,q)	(P,D,Q,s)	MAE
(0,0,1)	(1,1,1,52)	145856
(0,0,0)	(1,1,1,52)	145916
(0,0,2)	(0,1,1,52)	146910
(0,0,1)	(0,1,0,52)	163901
(2,0,2)	(1,1,0,52)	172505

Table 5.17: Top 5 SARIMA models based on MAE for week credit

(p,d,q)	(P,D,Q,s)	MAE
(1,1,0)	(0,1,0,52)	201674
(2,0,0)	(1,1,1,52)	207894
(2,1,0)	(1,0,0,52)	208050
(2,0,0)	(0,1,1,52)	211159
(2,0,0)	(1,1,0,52)	213846

Table 5.18: Top 5 SARIMA models based on MAE for week debit

(p,d,q)	(P,D,Q,s)	MAE
(1,1,1)	(0,1,0,52)	284537
(2,0,0)	(0,1,0,52)	284729
(0,1,2)	(0,1,0,52)	285043
(2,0,0)	(1,1,0,52)	286750
(1,0,2)	(1,1,0,52)	288027

Table 5.19: Top 5 SARIMA models based on MAE for week cashflow

(p,d,q)	(P,D,Q,s)	MAE
(0,0,0)	(0,1,1,12)	587550
(0,0,1)	(0,1,0,12)	618124
(0,0,1)	(1,1,0,12)	626565
(1,0,0)	(1,1,1,12)	645900
(2,0,0)	(0,1,1,12)	667860

Table 5.20: Top 5 SARIMA models based on MAE for month credit

(p,d,q)	(P,D,Q,s)	MAE
(0,0,1)	(0,1,1,12)	367043
(2,0,0)	(0,1,1,12)	367058
(2,0,0)	(1,1,1,12)	367572
(0,0,1)	(1,1,1,12)	368321
(1,0,2)	(0,1,1,12)	373207

Table 5.21: Top 5 SARIMA models based on MAE for month debit

(p,d,q)	(P,D,Q,s)	MAE
(2,1,1)	(1,1,0,12)	344779
(1,0,2)	(1,1,0,12)	345564
(0,0,2)	(0,0,1,12)	352422
(2,0,0)	(1,0,1,12)	353507
(0,0,1)	(1,1,1,12)	354610

Table 5.22: Top 5 SARIMA models based on MAE for month cashflow

Conclusion SARIMA

Tables 5.23 and 5.24 show the best SARIMA models based on RMSE and MAE respectively for each time series. To see what the best method is to forecast the cash flow with SARIMA, the evaluations of the cash flow and combined cash flow are compared. For RMSE this is shown in Table 5.25. Week cashflow is the forecast created by the best performing SARIMA model for forecasting weekly cashflow SARIMA(1,0,2)(0,1,0,52). Week combined is a forecast created by combining the best forecasts of Credit week and Debit week. Month cashflow and Month combined are made in a similar matter. For weekly forecasts, it depends on the criteria of whether the original forecast or the combined forecast is better. For monthly data, however, the original forecast performs better than the combined forecast.

For MAE this is shown in Table 5.26. Week cashflow is the forecast created by the best performing SARIMA model for forecasting weekly cashflow SARIMA(2,1,1) (1,1,0,12). Week combined is a forecast created by combining the best forecasts of Credit week and Debit week. Month cashflow and Month combined are made in a similar matter. For weekly forecasts, it depends on the criteria of whether the original forecast or the combined forecast is better. For monthly data, however, the original forecast performs better than the combined forecast.

Time Series	**(p,d,q)**	**(P,D,Q,s)**	**RMSE**
Week credit	(0,0,1)	(1,1,1,52)	210240
Week debtor	(2,1,2)	(1,0,1,52)	406033
Week cashflow	(1,0,2)	(0,1,0,52)	440998
Month credit	(0,0,2)	(0,1,0,12)	686409
Month debit	(1,0,2)	(0,1,1,12)	481184
Month Cashflow	(2,0,0)	(1,0,1,12)	419787

Table 5.23: Best performing SARIMA model based on RMSE for each time series

Time Series	**(p,d,q)**	**(P,D,Q,s)**	**MAE**
Week credit	(0,0,1)	(1,1,1,52)	145856
Week debtor	(1,1,0)	(0,1,0,52)	201674
Week cashflow	(1,1,1)	(0,1,0,52)	284537
Month credit	(0,0,0)	(0,1,1,12)	587550
Month debit	(0,0,1)	(0,1,1,12)	367043
Month Cashflow	(2,1,1)	(1,1,0,12)	344779

Table 5.24: Best performing SARIMA model based on MAE for each time series

Name	RMSE	MAE	MAPE
Week cashflow	440998	284149	97
Week combined	439729	316163	89
Month cashflow	419787	353507	134
Month combined	489105	397124	191

Table 5.25: Performance of SARIMA Forecasting Models

Name	RMSE	MAE	MAPE
Week cashflow	441812	284537	94
Week combined	441348	289732	97
Month cashflow	428250	344779	152
Month combined	464371	369625	167

Table 5.26: Performance of SARIMA Forecasting Models

5.2.3 LSTM

Parameter Tuning

As stated earlier in this thesis a number of parameters need to be tuned. The tests are done in the order of the tables. The results are taken into account when testing the next parameter. The LSTM models are tuned with RMSE and MAE. These results can be found in tables 5.27-5.30 and 5.31-5.34 respectively. In table 5.27 the number of hidden layers is tested. There are three time series that are better off with one hidden layer and three-time series that are better off with two hidden layers. The next parameter that is tested is the number of neurons. Note that the networks have different amounts of layers based on previous results. In Table 5.28 it is shown that for a four-time series, it was best to have 50 neurons and for a two-time series it was best to have 100 neurons. In table 5.29 the RMSE for the amount of epochs is shown. For three series it was best to have 100 epochs, for 2 series it was best to have 200 epochs and for one it was best to have 400 epochs. Table 5.30 shows the optimal batch size for each series. For three series it was best to have a batch size of 32 and for three series it was best to have a batch size of 64. The optimal parameters for each series are shown in Table 5.35

In table 5.31 the number of hidden layers is tested. There are four time series that are better off with one hidden layer and two-time series that are better off with two hidden layers. The next parameter that is tested is the number of neurons. In Table 5.32 it is shown that for a four-time series, it was best to have 50 neurons and for a two-time series it was best to have 100 neurons. In Table 5.33 the MAE for the amount of epochs is shown. For one series it was best to have 100 epochs, it was also for one series to have 200 epochs. It was for three series best to have 300 epochs and for one series it was best to have 400 epochs. Table 5.34 shows the optimal batch size for each series. For the series it was best to have a batch size of 64, for the 2 series it

was best to have a batch size of 16 and for one series it was best to have a batch size of 32. The optimal parameters for each series are shown in Table 36.

Hidden Layers	**RMSE One**	**RMSE Two**
Credit Week	252727	321641
Credit Month	306380	383043
Debit Week	298357	293873
Debit Month	921471	959913
Cashflow Week	360777	342625
Cashflow Month	588315	400393

Table 5.27: RMSE Comparison between One and Two Hidden Layer Models

Number of Neurons	**RMSE 50**	**RMSE 100**	**RMSE 200**
Credit Week	175779	208411	245978
Credit Month	325913	312039	325638
Debit Week	270874	284688	341604
Debit Month	773283	941981	928449
Cashflow Week	325240	350755	433990
Cashflow Month	496024	445699	488636

Table 5.28: RMSE Comparison for Different Neuron Counts in LSTM Layers

Epochs	**RMSE 100**	**RMSE 200**	**RMSE 300**	**RMSE 400**
Credit Week	164400	186697	168887	179048
Credit Month	256203	270093	318336	327688
Debit Week	293779	312259	298709	287951
Debit Month	628284	767364	812935	785650
Cashflow Week	311327	307685	323296	338746
Cashflow Month	356956	353670	360261	495338

Table 5.29: RMSE Comparison for Different Epoch Counts in LSTM Training

Batch Size	**RMSE 16**	**RMSE 32**	**RMSE 64**
Credit Week	179553	171708	168118
Credit Month	286519	276629	276743
Debit Week	274439	270049	296187
Debit Month	644918	662898	627370
Cashflow Week	368267	287925	359275
Cashflow Month	466028	390295	359766

Table 5.30: RMSE Comparison for Different Batch Sizes in LSTM Training

Hidden Layers	MAE One	MAE Two
Credit Week	172818	218358
Credit Month	228445	225828
Debit Week	171188	184066
Debit Month	625511	863648
Cashflow Week	229085	267985
Cashflow Month	556606	324027

Table 5.31: MAE Comparison between One and Two Hidden Layer Models

Number of Neurons	MAE 50	MAE 100	MAE 200
Credit Week	128482	213060	202053
Credit Month	206857	260960	265782
Debit Week	181397	185969	182991
Debit Month	674449	658136	677917
Cashflow Week	223051	215981	238169
Cashflow Month	350880	394461	384322

Table 5.32: MAE Comparison for Different Neuron Counts in LSTM Layers

Epochs	MAE 100	MAE 200	MAE 300	MAE 400
Credit Week	141141	121423	144874	157472
Credit Month	188436	186573	174856	174980
Debit Week	293779	312259	298709	287951
Debit Month	617112	637988	682771	556098
Cashflow Week	224421	238005	236401	237323
Cashflow Month	320186	416336	318175	366009

Table 5.33: MAE Comparison for Different Epoch Counts in LSTM Training

Batch Size	MAE 16	MAE 32	MAE 64
Credit Week	168574	144040	130167
Credit Month	247420	237021	201657
Debit Week	187706	183732	182121
Debit Month	654702	753982	679713
Cashflow Week	227673	230074	237885
Cashflow Month	414755	358230	363975

Table 5.34: MAE Comparison for Different Batch Sizes in LSTM Training

Conclusion LSTM

To see what the best methods are to forecast the cash flow with LSTM concerning RMSE and MAE, the evaluations of the cash flow and combined cash flow are compared. This is shown in table 5.35 and 5.36. Week cash flow is a forecast created by an LSTM with the parameters that were best for forecasting weekly cash flow. Week combined is a forecast created by combining the forecasts of Credit week and Debit week. Month cashflow and Month combined are made in a similar matter. According to table 5.35, it is best not to combine the forecasts of debit and credit week, but use the cashflow forecast. According to table 5.36, it is better for weekly forecasts to use the forecast created by combining the forecasts of credit week and debit week. For monthly data, however, we can see that the original forecast performs better than the combined forecast.

Parameters	**layers**	**Neurons**	**Epochs**	**Batch Size**	**RMSE**
Credit Week	1	50	100	64	168118
Credit Month	1	100	100	32	276629
Debit Week	2	100	400	32	270049
Debit Month	1	50	100	64	627370
Cashflow Week	2	50	200	32	287925
Cashflow Month	2	100	200	64	359766

Table 5.35: optimal LSTM parameters for time series

Parameters	**layers**	**Neurons**	**Epochs**	**Batch Size**	**MAE**
Credit Week	1	50	200	64	130167
Credit Month	1	50	300	64	201657
Debit Week	2	50	300	64	182121
Debit Month	1	100	400	16	654702
Cashflow Week	2	100	100	16	227673
Cashflow Month	2	50	300	32	358230

Table 5.36: optimal LSTM parameters for time series

Name	**RMSE**	**MAE**	**MAPE**
Week cashflow	287925	229722	191
Week combined	290121	221365	130
Month cashflow	359766	319221	98
Month combined	530213	442580	162

Table 5.37: Performance of LSTM Forecasting Models

Name	**RMSE**	**MAE**	**MAPE**
Week cashflow	298416	227673	166
Week combined	283316	212057	140
Month cashflow	456788	358230	198
Month combined	867921	678392	230

Table 5.38: Performance of LSTM Forecasting Models

5.3 Final result

In Table 5.39 and Table 3.40 the best models for each method are used to minimize the RMSE and MAE respectively.

In Table 5.39, LSTM outperforms ETS and ARIMA for every time series except for Debtor Monthly. Since the best-performing forecasts for weekly and monthly cash flow in the LSTM method was done on cash flow data, these are the best ways to forecast the weekly and monthly cash flow overall. The forecasts can be seen in Figures 5.1-5.3

In Table 5.40, LSTM outperforms ETS and ARIMA for every time series except for Debtor Monthly and Cashflow Monthly. Since the best-performing forecast for weekly cash flow in the LSTM method was determined to be the combined forecast of Credit week and Debit week, this is the best way to forecast the weekly cash flow overall. For the monthly forecast, the best forecast came from a SARIMA network that forecasts cash flow every month.

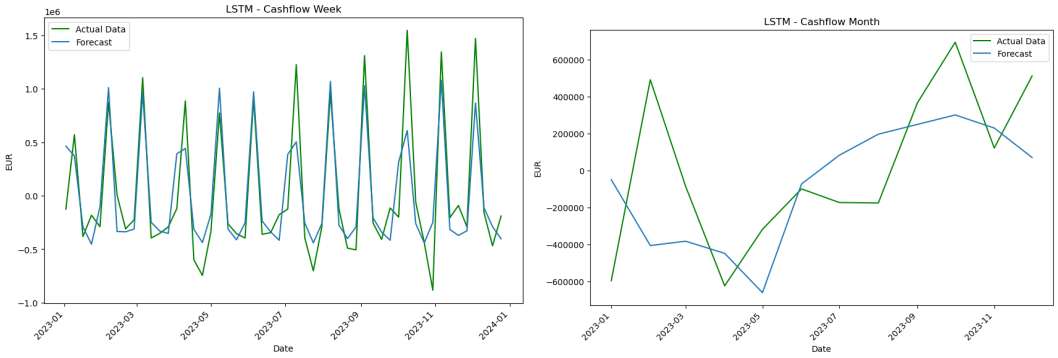
Both tables 5.39 and 5.40 show that LSTM does not work well for monthly debtor transactions.

Time Series	**ETS RMSE**	**SARIMA RMSE**	**LSTM RMSE**
Cashflow Monthly	455463	419787	359766
Cashflow Weekly	432398	439729	287925
Creditor Monthly	629809	686409	276629
Creditor Weekly	226574	210240	168118
Debtor Monthly	483639	481184	627370
Debtor Weekly	414215	406033	270049

Table 5.39: RMSE for Best Performing Models for Each Time Series (ETS, ARIMA, and LSTM)

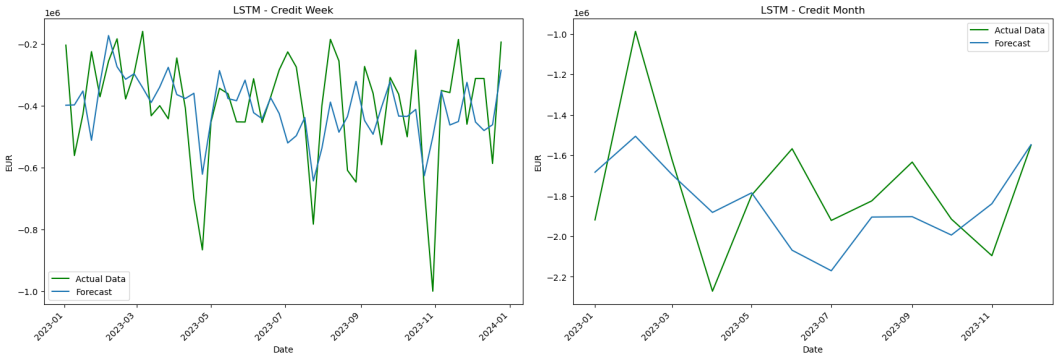
Time Series	**ETS MAE**	**SARIMA MAE**	**LSTM MAE**
Cashflow Monthly	363908	344779	358230
Cashflow Weekly	278063	284537	212057
Creditor Monthly	584452	587550	201657
Creditor Weekly	166455	145856	130167
Debtor Monthly	339517	367043	654702
Debtor Weekly	215407	201674	182121

Table 5.40: MAE Comparison of Best Performing Models for Each Time Series



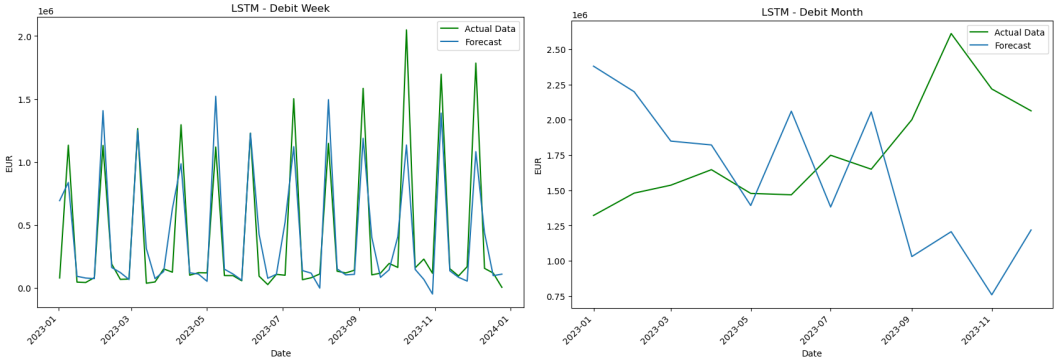
(a) Weekly Cashflow (b) Monthly Cashflow

Figure 5.1: Best LSTM model for Cashflow



(a) Weekly Transactions (b) Monthly Transactions

Figure 5.2: Best LSTM model for Creditor Transactions



(a) Weekly Transactions (b) Monthly Transactions

Figure 5.3: Best LSTM model for Debtor Transactions

Chapter 6

Conclusion

This research set out to explore the best methods for forecasting cash flow for GS Interfer, comparing the accuracy of machine learning techniques.

The methods were tuned to minimize the root mean squared error and the mean absolute error. The results show that LSTM outperformed all others for weekly and monthly data in terms of RMSE. LSTM also outperformed the others in weekly forecasts in terms of MAE. For monthly cashflow regarding MAE, SARIMA had the best forecast.

These results highlight the importance of choosing the right forecasting model and demonstrate that machine learning methods can significantly improve forecasting accuracy for cash flow management.

One limitation of this study is that it does not account for external factors such as market trends, which could influence cash flow. Additionally, the scope of the data was limited to historical debtor and creditor transactions, without considering broader financial variables.

Further research could integrate external economic indicators into the forecasting models or test hybrid models that combine both traditional and machine learning techniques to enhance predictive power.

Overall, this thesis shows that adopting machine learning methods, particularly LSTM models, can lead to more accurate and reliable cash flow forecasts, ultimately helping businesses optimize their financial strategies and decision-making processes.

Chapter 7

Bibliography

Brownlee, J. (2019). Deep Learning for Time Series Forecasting in Finance. *arXiv preprint arXiv:1902.10877*.

Efron, B., & Hastie, T. (2021). *Computer Age Statistical Inference, Student Edition: Algorithms, Evidence, and Data Science*. Cambridge University Press.

Fischer, T., & Krauss, C. (2018). Replicating a Trading Strategy with Machine Learning Techniques. *Quantitative Finance*, 19(4), 1-25.

Gardner, E. S., & McKenzie, E. (1985). Forecasting trends in time series. *Management Science*, 31(10), 1237–1246.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8), 1735-1780.

Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep neural network training by reducing internal covariate shift. *Proceedings of The 32nd International Conference on Machine Learning (ICML 2015)*, 448–456.

Keskar, N. S., Nocedal, J., Mudigere, D., Nadir, M., Shi, P., & Tang, H. (2017). On large-batch training for deep learning: Generalization gap and sharp minima. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, PMLR 70:1725-1734.

Kwiatkowski, D., Phillips, P. C. B., Schmidt, P., & Shin, Y. (1992). Testing the null hypothesis of stationarity against the alternative of a unit root: How sure are we that economic time series have a unit root? *Journal of Econometrics*, 54(1-3), 159–178.

Mishra, P. K. (2016). Forecasting Time Series Data: ARIMA vs. Machine Learning Approaches. *Theoretical and Applied Economics*, 23(4), 19-32.

Singh, V., & Tripathi, M. (2014). Comparative Study of Machine Learning Techniques for Stock Price Prediction. *Proceedings of the 16th UKSIM International Conference on Computer Modelling and Simulation*.

Stathakis, D. (2009). How many hidden layers and nodes? *International Journal of Remote Sensing*, 30(8), 2133–2147.

Zhang, L., Tang, L., Gao, H., & Pan, R. (2018). Machine Learning-Based Financial Trading Strategy Optimization Using LSTM Neural Networks. *arXiv preprint arXiv:1803.06386*.