Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Sindre J. I. Sivertsen
Sander K. Kilen

Exploring a Convolutional Autoencoder with LSTM on E-commerce Time-Series Forecasting

Master's thesis in Datateknologi
Supervisor: Anders Kofod-Petersen
June 2022

NTNU
Norwegian University of Science and Technology

Sindre J. I. Sivertsen
Sander K. Kilen

Exploring a Convolutional Autoencoder with LSTM on E-commerce Time-Series Forecasting

Master's thesis in Datateknologi
Supervisor: Anders Kofod-Petersen
June 2022

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

NTNU
Norwegian University of
Science and Technology

Abstract

This thesis explores the use of a convolutional autoencoder and LSTM model for making time series forecasts on product category trend data supplied by "Prisguiden.no". The use of the CNN-AE-LSTM model is expanded by applying a local univariate, global univariate, local multivariate and global multivariate model. Results are compared with a LSTM baseline model applying the same model types as the CNN-AE-LSTM model. Additionally, a SARIMA model is used as a statistical baseline for forecasting.

The experiment results show that with the E-commerce data from "Prisguiden.no", the local multivariate LSTM model is the most accurate. The results indicate that the CNN-AE-LSTM performance is conditionally dependent on datasets with high levels of noise in order to outperform the LSTM. The model achieves a small performance increase on datasets with high noise, but predictions will suffer on data with low levels of noise.

The CNN-AE-LSTM model is not well suited for applications with the use of trend data from "Prisguiden.no", while the local multivariate LSTM is the model best suited for such predictions.

Sammendrag

Denne masteroppgaven utforsker bruk av en convolutions-autoencoder og LSTM modell for å gjennomføre tidsserie prediksjon av produkt kategori trend data fra "Prisguiden.no". Denne masteroppgaven arbeider mot å utvide den teoretiske kunnskapen om bruk av denne CNN-AE-LSTM modellen ved å lage modeller som er både lokale og globale, samt ved bruk av en univariabel og multivariabel modell. Resultatene fra disse eksperimentene er sammenlignet med resultater fra LSTM modeller av samme type, globale og lokale, univariable og multivariabel modeller. I tillegg er disse modellene sammenlignet med en statistisk "baseline" ved bruk av den statistiske modellen SARIMA.

Resultatene fra disse eksperimentene viser at bruk av en lokal multivariabel LSTM modell er det som er best egnet for å gjennomføre prediksjoner på dataen fra "Prisguiden.no". Eksperimentene indikerer at CNN-AE-LSTM modellene er sterkt avhengig av type data som som skal predikeres, og er spesielt egnet til bruk på data med store mengder støy. Ved bruk av et datasett med mye støy indikerer eksperiment resultatene at CNN-AE-LSTM modellene utkonkurrerer LSTM modellen. CNN-AE-LSTM modellen gjør det hakket bedre på data med mye støy, men er svært mye dårligere enn LSTM modellen på datasett med lite eller ingen datastøy.

CNN-AE-LSTM modellen er ikke velegnet for bruk til tidsserie prediksjoner på data fra "Prisguiden.no". En lokal multivariabel LSTM modell er derimot bedre egnet for slike prediksjoner.

Preface

This work is done as part of the author's masters thesis, and is an extension to the work done in Sivertsen and Kilen [2021].

Some sections of this thesis have earlier been submitted and graded as part of the fall project in Sivertsen and Kilen [2021]. Others have been improved and extensively altered as part of this thesis.

Previously written sections consist of large parts of *Background and Theory [Chapter 2]*, and *Related work [Chapter 3]*, and Section 5.1. However, multiple sections are updated and added to the chapters. To avoid misconceptions we have marked chapters that contain no new information with *.

Additionally, multiple sections have been extensively updated as part of this masters thesis. These include *Introduction [Chapter 1]*, as well as Section 4.3.

All of the code are open and available at [Github Sindre Sivertsen [2022]]. The dataset used is not available.

Acknowledgements

We wish to thank our supervisor Anders Kofod-Petersen for his help and valuable feedback during this project. Additionally, we would like to thank "Prisguiden.no" for their help and cooperation, as well as for the dataset and this intriguing project.

Sindre J.I Sivertsen & Sander K Kilen
Trondheim, June 7, 2022

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Section 1.1 introduces the underlying problem of time series prediction in an E-commerce setting and establishes the motivation behind this thesis. Section 1.2 presents the problem at hand. Section 1.3 lists the goal and relevant research questions of the thesis, followed by the research method presented in Section 1.4. The contributions made in this thesis are presented in Section 1.5. Lastly, the thesis structure is presented in Section 1.6.

1.1 Background and Motivation

With the emergence of the internet, large parts of the human experience are conducted online. Online services supply users with everything from entertainment and social media to banking and online shopping. The accessibility of online services such as online retailers has enabled users to shop for most products online. With online shopping, competitive pricing emerged. When shopping online, a user will often consider the product's pricing before completing a purchase, often comparing the retail prices between different retailers.

In order to easily and effectively compare prices of products amongst retailers, *"Prisguiden.no"* was created. Products and product categories were introduced to the site, collecting the pricing information of the products from multiple different retailers. When new products hit the market, these products are introduced to the product portfolio in order for users to compare the prices. Operating such an online service has enabled Prisguden to accumulate user data such as product and product category interests.

The user interests data is currently unused, despite being collected for years. The collected time series data could hold relevant information. Prisguiden intends to use this data in order to predict future product and category trends.

With this information, the allocation of resources used for product category updates could be more informed as to what is most relevant.

In order to make such time series predictions, methods for time series forecasting are used. Using methods to evaluate historical data in an attempt to predict future values. Methods have therefore been introduced to achieve this.

1.2 Problem description

Online shopping platforms retain large amounts of product sales and interest data. Despite this, it is not always easy to know what information this data might contain and what it could mean to exploit and analyze this data. Analyzing product interest data or sales data could help retailers discover helpful information such as product trends or anomalies. While methods for attaining such information already exist to some degree, in the form of simple statistical methods or neural networks, there is still room for improvement.

This thesis will focus on proposing a solution for predicting product trends on time series data with multiple product categories.

1.3 Goals and Research Questions

This thesis explores using machine-learning algorithms to accurately forecast the user interest in product categories on a price comparison website. Exploiting the predictive abilities of deep Neural Networks such as Convolutional Neural Networks, Long-Short term memory, and Autoencoders, we intend to introduce a new predictive model into the problem space. The intention is to outperform the current state-of-the-art predictive algorithms in the current domain.

The data supplied by "Prisguiden.no" contains historical data about user activity, such as visitation and click data. We intend to use this data to predict future product and category trends based on the historical data.

Goal *To accuratly predict future product category trends based on historical visitation and click data using a Convolutional Autoencoder with LSTM.*

To measure the viability and accuracy of the proposed goal, we need to look into the already proposed methods in this problem space. This is required in order to assess whether or not our goal is rendered mute due to previous solutions.

RQ1 *What are the existing solutions for predicting future product category trends, or sales trends, based on historical time series data?*

RQ2 *How does the different solutions found by addressing RQ1 compare to each other?*

RQ3 *What is lacking in the current solutions found by addressing RQ1, and how could these be improved upon?*

The results of the Structured literature review are required for the reader to understand the context. The literature addressing RQ1, RQ2, and RQ3 have earlier been submitted and graded as part of the fall project in [Sivertsen and Kilen, 2021]. It is included here for completeness, but the conclusion of these can be found in [Sivertsen and Kilen, 2021].

RQ4 *How will a baseline LSTM compare against SARIMA using error metrics MASE and sMAPE?*

RQ4.1 *How will different LSTM model structures affect its results? E.g. a global univariate or a local multivariate model.*

RQ5 *Can a Convolutional Autoencoder and LSTM model achive higher predictive accuracy than current state of the art models?*

The proposed model is compared to baselines from the current state-of-the-art methods used to evaluate the validity of the proposed model.

1.4 Research Method

Through this thesis, the goal and research questions defined in section 1.3 is approached in two different ways.

Initially, research question 1 through 3 is addressed through a theoretical analysis of current literature connected to the subject. Current state-of-the-art methods and frameworks are compared to assess current valid solutions to the problem space. Additionally, current work done within the E-commerce sector attempting to predict sales and trends is reviewed. We argue that this problem space is similar to our proposed problem, and we then use this as a benchmark in order to assess new solutions. The literature review aims at reviewing the current state of time series prediction in order to find a method to achieve our goal described in RQ1-RQ3.

Secondly, research questions 4 and 5 are assessed through practical experimentation. The proposed convolutional autoencoder and LSTM are tested on the available dataset in order to make predictions. Predictions made by the model can then be compared to predictions done by baselines models on the same dataset. The baselines are created as a result of the literature search conducted in order to find other current state-of-the-art models. With the baseline models created, the CNN-AE and LSTM model can be evaluated to assert if

the model achieves higher predictive accuracy than other models. In addition to the proposed CNN-AE-LSTM model, different model structures are tested and compared to the baseline.

1.5 Contributions

The main focus of this thesis is to assess current state-of-the-art time series prediction in E-commerce forecasting, and make a comparison with a new state-of-the-art predictive method. Contributing to state-of-the-art, we propose the use of a new predictive method in E-commerce forecasting. Introducing a predictive method that has yet to be applied in an E-commerce setting.

The main contributions of this thesis are:

1. *To evaluate and compare current methods of time series prediction on E-commerce forecasting.*
2. *To formulate a framework for achieving higher predictive accuracy than the current state-of-the-art methods on our problem space.*
3. *To compare current state-of-the-art methods against a new predictive model to evaluate predictive ability.*
4. *To develop guidelines for forecasting on category trends on "Prisguiden.no".*

1.6 Thesis Structure

This thesis introduces a lot of different theories and subjects, thus a clear structure is required. Chapter 2 introduces the underlying theory of the most important concepts presented in this thesis, ranging from time series prediction to deep learning methods and concepts. Chapter 3 introduces work related to our problem-space, creating the theoretical basis for achieving our proposed goal and research questions. This chapter focuses on current state-of-the-art methods, as well as new proposed frameworks and methods relevant to our solution. Chapter 5 discusses the available data for the project, presenting data analysis, filtering and pre-processing. Afterward, chapter 5 describes the proposed model for this project, as well as model designs for creating baselines. Chapter 6 describes the methodology used when conducting the experiments and implementation of the model and baselines, before Chapter 7 presents the results achieved after the method implementation. Lastly, Chapter 8 contains the discussion part of the thesis, discussing the model selection, used methodology, results, and possible sources for errors, before it concludes and summarizes the key findings in the thesis.

Chapter 2

Background Theory and Motivation

This section will primarily cover the required background theory for this thesis. Initially, Section 2.1 covers the required theoretical basis for the rest of this thesis, introducing the required theory regarding time series and predictive models. Section 2.2 covers the Structured literature review conducted concerning the research questions proposed in this thesis, before Section 2.3 covers the underlying motivation for the work done in this thesis.

2.1 Background Theory

2.1.1 Time Series*

> A time series is a sequence of data points that occur in successive order over some period of time.

Kenton [2020]

In a time series, time is often the independent variable. Examples of time series are weather data, stock markets, sound level samples. The time t usually ranges over a discrete index set and is often equally spaced.

Properties

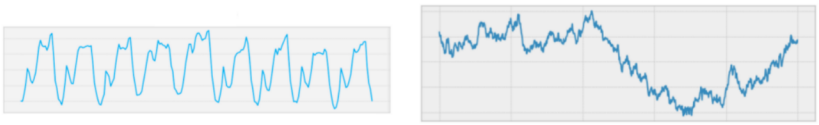
A time series has several properties:

Stationarity A time series is stationary if its statistical properties do not change over time. In other words, if it has a variance, mean, and covariance independent of time.

Rob J Hyndman [2014] defines stationarity more formally in Definition 1.

Definition 1 *X_t is a stationary time series $x_1, ..., x_n, if \forall_s \in \mathbb{R}$: the distribution of $(x_t, ..., x_{t+s})$ is equal*

Figure 2.1: Examples of stationarity



(a) A stationary Time-Series (b) Non-stationary Time-Series

Seasonality If the time-series follows periodic fluctuations, like how electricity usage varies during 24 hours, then it has seasonality. Figure 2.1a shows a clear cycle where a clear pattern repeats over again.

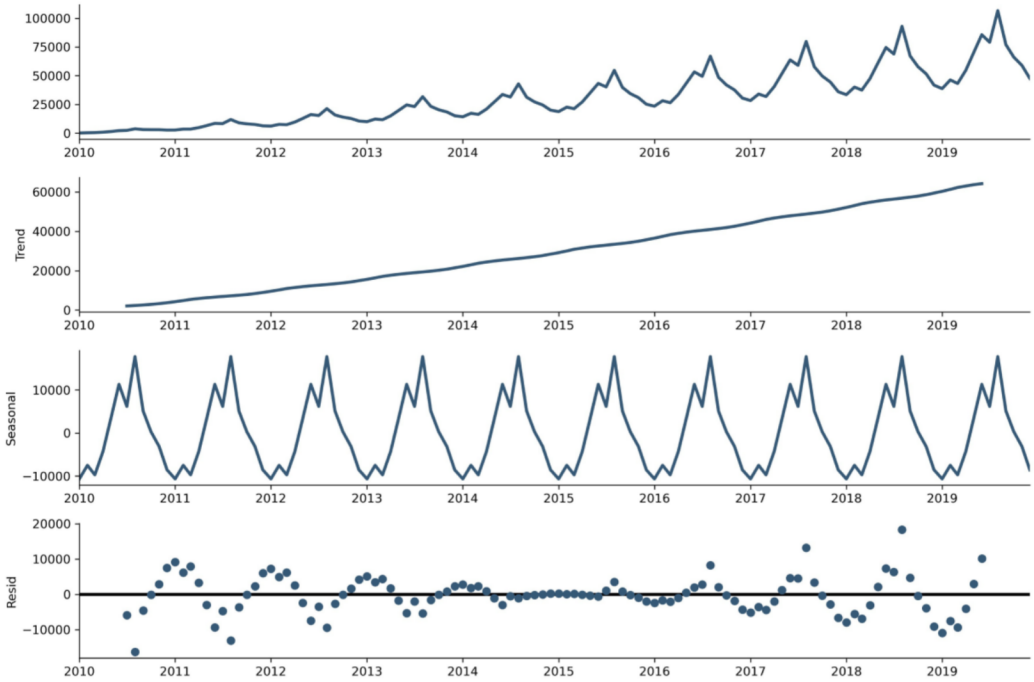
Autocorrelation If a time-series has a strong autocorrelation, then there is a big correlation between observations with a time lag between them.

Trends When a time-series has a deterministic component proportionate to the time period it has a trend. In simpler terms, if a time-series plot seems to center around an increasing or decreasing line, it suggests the presence of a trend. Figure 2.2 show a series with a clear growing trend.

Cycles Cycles differ from seasonality because the period does not have to be fixed.

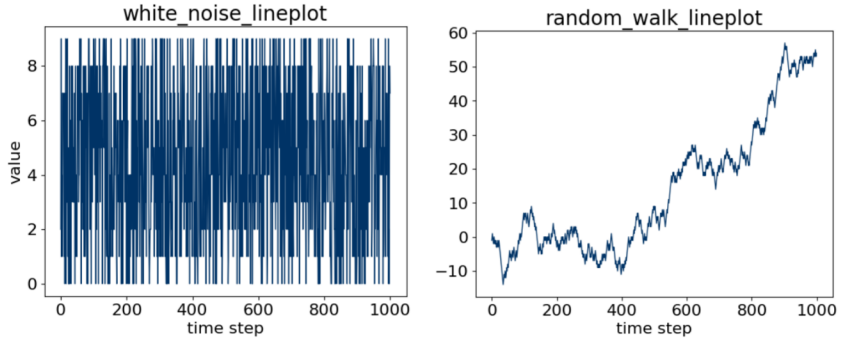
Level The level of a time series is equal to the mean. If a time series has a trend then the level is changing.

Figure 2.2: A Time-Series decomposed into its trend, seasonal, and residual parts



2.1.2 Random Walk

Figure 2.3: Syntetic data, noise and random values.



(a) White noise lineplot (b) A Random walk lineplot

White noise is just a random sample of numbers not following any pattern, as seen in Figure 2.3b.

A random walk is different from white noise, because the next value in the series is dependent on the previous value plus some noise.

$$y_t + 1 = y_t + r \tag{2.1}$$

Equation (2.1), where r is some random number, shows the equation for creating a random walk. An example of a random walk graph is shown in Figure 2.3b.

Figure 2.3b.

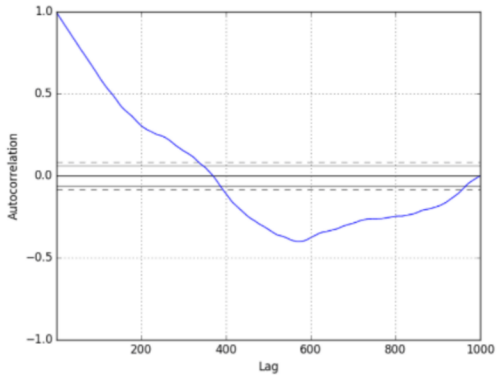


Figure 2.4: A Random walk autocorrelation

Since a random walk is highly dependable on previous values, this will clearly show in an autocorrelation plot. Autocorrelation plots illustrate how much a series correlates with its previous values. Figure 2.4 shows how a autocorrelation plot for the random walk in Figure 2.3b. It is a steadily decreasing trend that follows a linear pattern in the first 500 days. Figure 2.6 illustrates a random walk decomposed into trend, seasonality, and residual values.

One way to show if a series follows a random walk is to remove the temporal dependence by subtracting each value in the series by the previous value. This will leave only the noise r in Equation (2.1). If the series follows a random walk it will look a lot like the white noise shown in Figure 2.3a.

Plotting the autocorrelation of the remainding noise r in Figure 2.5 we can see the correlations are small, close to zero and below the 95% (vertical dotted line) and the 99% (vertical full line) confidence levels.

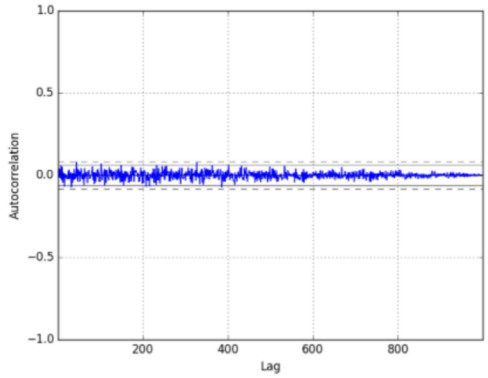


Figure 2.5: A Random walk noise autocorrelatoin

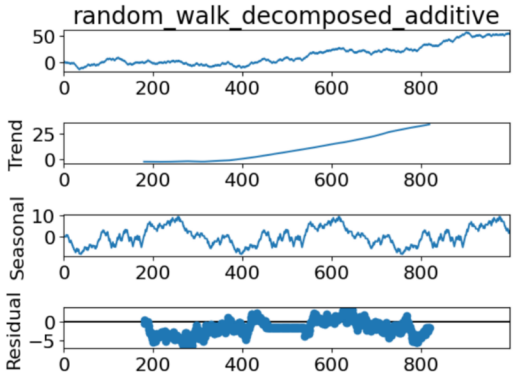


Figure 2.6: A Random walk decomposed in Trend, Season, and rest

There are multiple characteristics that identify a random walk.

- The time series shows a strong temporal dependence that decays linearly or in a similar pattern.
- The time series is non-stationary and making it stationary shows no obvious learnable structure in the data.
- The persistence model provides the best source of reliable predictions.

2.1.3 Forecasting time-series*

Let $Y = \{y_1, y_2, ..., y_n\}$ denote a time-series. Forecasting is predicting the next time step y_{n+h} where h is the forecasting horizon.

There are two main categories within time series forecsasting, **univariate** and **multivariate**. A univariate time series prediction consists of only one value or observation over a time period. Methods for prediction using univariate time series is called a univariate time series model. The model uses only one value series as the input sequence, making predictions solely based on the historical data. A multivariate time series is a set of multiple values spanning a period of time. A model that takes multiple time-dependent variables as input is called a multivariate model. These models evaluate the input sequences in relation to each other, as well as their historical values, in order to make predictions. Multivariate forecasting can result in the prediction of values for one of the observations or multiple observations.

Predictive models attempt to forecast values either as a single-step or multiple steps forward in time. Due to this, forecasting is categorized as *Single step prediction* and *Multi step predition*. **Single step prediction** forecasts values only one time step forward in time. **Multi-step prediction** forecasting values multiple time steps forward in time at once. Despite the difference, multi-step forecasting can be accomplished using single-step forecasting. By forecasting a single step forward in time several times, a single-step forecasting method can recursively accomplish multi-step prediction.

Assuming a stationary time series, several approaches to time series modeling are available. A naive method is through the use of mean values, predicting the next value to be the mean of all past observations. Additionally, a smaller subset of past observations can be used, applying a moving average across the time series. Longer subsets result in a smoother prediction graph.

Another viable prediction technique is **exponential smoothing**. It uses the same approach as the moving average method but differs through the use of a decreasing weight assigned to each observation.

$$y = \alpha x_t + (1 - \alpha) y_{y-1}, t > 0 \tag{2.2}$$

Equation (2.2) shows exponential smoothing, where α smoothing factor that takes values between 0 and 1. It determines how fast the weight decreases with time.

2.1.4 ARMA*

Auto-Regressive Moving Average **ARMA** is a statistical model for time series prediction. It is one of the most commonly used methods for univariate time series forecasting. $ARMA(p, q)$ is defined for stationary data and consists of two components $AR(p)$ and $MA(q)$.

The $AR(p)$ model is built on the assumption that the value of a given time series y_n can be estimated using a linear combination of the p past observations,

an error term ϵ_n and a constant term c as seen in Equation (2.3) [Ziegel et al., 1995].

$$y_n = c + \sum_{i=1}^{p} \phi_i y_{n-1} + \epsilon_n \tag{2.3}$$

where $\phi_i, \forall i \in \{1, ..., p\}$ denote the model parameters, and p is the order of the model.

The second part $MA(q)$ uses the past errors in a similar fasion Equation (2.4).

$$y_n = \mu + \sum_{i=1}^{q} \theta_i \epsilon_{n-1} + \epsilon_n \tag{2.4}$$

Here μ represents the mean of observations. q is the order of the model. $\theta_i, \forall i \in \{1, ..., q\}$ represents the parameters of the model.

Combining the past observations Equation (2.3) and past error terms Equation (2.4) we get the $ARMA(p, q)$ model in Equation (2.5).

$$y_n = c + \sum_{i=1}^{p} \phi_i y_{n-1} + \epsilon_n + \mu + \sum_{i=1}^{q} \theta_i \epsilon_{n-1} + \epsilon_n \tag{2.5}$$

2.1.5 ARIMA

The ARIMA model is an extension to the ARMA model, and works as a more generalized model. ARIMA models differ from ARMA models in their ability to transform a non-stationary time series into a stationary one.

The first part of ARIMA is the autoregression model $AR(p)$ where p is the maximum lag.

The second part is the moving average model $MA(q)$ where q is the maximum lag.

The third part is the order of integration $I(d)$ where d is the number of differences required to make the series stationary.

2.1.6 SARIMA

In addition to the ARIMA model, the SARIMA has been developed. The SARIMA model extends the ARIMA model by adding a seasonal component to the model, enabling the model to apply different transformations to a non-stationary seasonal time series in order to remove seasonality and non-stationary behaviors. [Utlaut, 2008, p. 327-385].

The SARIMA model extends the ARIMA by adding a final component for seasonality.

The final component is seasonality $S(P, D, Q, s)$, where s is the length of the season. s is dependent on P and Q, which are equal to p and q but for the seasonal component. D is the number of differences required to remove seasonality from the series.

The combination of all these parts is the SARIMA model, $SARIMA(p, d, q)(P, D, Q, s)$.

2.1.7 Loss functions*

Russel and Norvig defines loss functions as such:

> A *loss function* $L(x, y, \hat{y})$ is defined as the amount of utility lost by predicting $h(x) = \hat{y}$ when the correct answer is $f(x) = y$ and h is the heuristic function. This is the most general formulation of the loss function. Often a simplified version is used, $L(y, \hat{y})$, that is independent of x [Russel and Norvig, 2012, p. 710-711].

This means that the loss function is the function that calculates the error between the models prediction, and the actual target value. This chapter will briefly explain standard loss functions.

MSE

The most commonly used loss function for regression problems is the **Mean Squared Error (MSE)** function in Equation (2.6). It is the mathematically preferred function if the target distribution is Gaussian. It punishes large errors much more harshly than smaller errors due to its squaring of the error. Here $e = y - \hat{y}$, where y is the actual value and $\hat{y}$ is the predicted value.

$$MSE = \frac{1}{n}\sum_{t=1}^{n} e_t^2 \tag{2.6}$$

MAE

If the target distribution consists of outliers, then the **Mean Absolute Error (MAE)** in Equation (2.7) is more appropriate as it does not punish the outliers too much.

$$MSE = \frac{1}{n}\sum_{t=1}^{n} |e_t| \tag{2.7}$$

MAPE

The Mean Absolute Percentage Error Equation (2.8) (MAPE) is a popular metric for evaluating forecasting performance. y is the actual target value of the target we are trying to forecast, and $\hat{y}$ is the predicted value. t is the time index.

The advantages of MAPE is that it is expressed as a percentage, which means it is scale-independent and can be used for forecasting on different scales. A percentage is also easily explainable.

$$MAPE = \frac{1}{n}\sum_{t=1}^{n}\frac{|\hat{y}_t - y_t|}{(y_t)} \tag{2.8}$$

A significant shortcoming of MAPE is that it is undefined when the actual value y is 0. It will also produce extreme values if the value is close to 0.

MAPE is also asymmetric and puts a higher penalty on errors where the predicted $\hat{y}$ is higher than the actual value y. This is because as long as we are dealing with positive numbers, the highest bound for a low forecast is 100%. But there is no upper limit for forecasts that are too high. As a result, the error function will favor models that under-predict rather than over-predict a forecast.

SMAPE

Symmetric Mean Absolute Percentage Error (SMAPE) shown in Equation (2.9) is an error function made to overcome some of the shortcomings of the MAPE function. By incorporating $\hat{y}$ to the denominator, SMAPE is symmetrical, with a lower bound of 0% and an upper bound of 200%.

SMAPE is one of the most used performance measures and is used in many forecasting competitions.

SMAPE is still vulnerable to denominator values close to zero. Hewamalage et al. [2021] solves the zero problem by changing the denominator of the SMAPE to $max(|y| + |\hat{y}| + \epsilon, 0.5 + \epsilon)$, where ϵ is set to 0.1. This version of SMAPE avoids division by zero by switching to an alternate positive constant for the denominator when the forecasting values are too small.

$$SMAPE = \frac{1}{n}\sum_{t=1}^{n}\frac{|\hat{y}_t - y_t|}{(|y_t| + |\hat{y}_t|)/2} \tag{2.9}$$

$$SMAPE_{ALT} = \frac{1}{n}\sum_{t=1}^{n}\frac{|\hat{y}_t - y_t|}{max(|y| + |\hat{y}| + \epsilon, 0.5 + \epsilon)} \tag{2.10}$$

MASE

The Mean Absolute Scaled Error **MASE** proposed by Hyndman and Koehler [2006], is a scale-free error metric that compares predictions with the output of a Naive Forecast.

$$MASE = \frac{MAE}{MAE_{in-sample,naive}}$$

where MAE is the mean absolute error produced by the actual forecast and $MAE_{in-sample,naive}$ is the mean absolute error produced by a naive forecast, calculated on the in-sample data. The naive forecast is predicting that the next time step is equal to the actual value of the current time step.

$$MASE = \frac{\frac{1}{J}\sum_{j=1}^{n}|\hat{y_j} - y_j|}{\frac{1}{T-1}\sum_{t=2}^{n}|y_t - y_{t-1}|}$$

where

J = Number of forecasts

t = The training set

T = Number of samples in the training set

$$MASE = \frac{\overbrace{\frac{1}{J}\sum_{j=1}^{n}|\hat{y_j} - y_j|}^{\text{MAE}}}{\underbrace{\frac{1}{T-m}\sum_{t=m+1}^{n}|y_t - y_{t-m}|}_{MAE_{in-sample,naive}}} \tag{2.11}$$

where:

m = seasonal period

The main difference is the denominator is MAE of the one-step seasonal naive forecast method on the training set.

MASE is a scale-independent measure, which makes it a good choice for domains with multiple time series of different scales.

MASE is normalized by the average in-sample one-step seasonal forecast error. So a MASE value greater than 1 indicates that the model tested is worse compared to the naive benchmark. The closer the MASE error is to 0, the better the model.

2.1.8 Hyperparameter selection

Machine learning algorithms are heavily dependent on the selected hyperparameters for individual situations. There are several different approaches to hyperparameter tuning that are used within the field of machine learning. Some methods approach the problem with heavily exhaustive search methods, whilst others optimize the search space by selecting only a subspace of the search range for finding optimal parameters for a model.

Grid search

Grids search is an example of a hyperparameter tuning method relying on an exhaustive search of the parameter range. By trying out all possible combinations of parameters, the model is likely to find a combination that fits the model well for the current problem space. In order to evaluate the models fitted through the extensive search, some sort of performance metric is required in order to compare parameters. Such metrics can be measured through the use of methods such as cross-validation or through the use of a separate validation set.

In order to tune models, the range of parameters to be selected must be specified beforehand so that it is clear as to which range of parameters should be tuned. Géron [2017]

Randomized search

In cases where the number of parameter combinations are limited, grid search is a fitting approach. However, when the range of parameter combinations is large, it will result in high time consumption in order to fit all the different combinations. Other methods are therefore often more relevant. One such method is *Randomized search* or *Random search*. Randomized search reduces the parameter search space by selecting only a limited number of parameter combinations at random. In contrast to grid search where the range of values for each parameter is limited as to not create a too large parameter search space, random search does not share this problem. The random search method runs only a selected number of iterations before completing, disregarding the range of the search space. Thus the computational budget of the parameter search is determined by the number of iterations, not by the size of the parameter search space. Géron [2017]

Bayesian tuning

Although random search reduces the search space of grid search, the selection of parameters is only random. The next step is therefore to consider a more informed search of parameters. One such approach is to use *Bayesian Optimization* or *Bayesian search*.

Bayesian optimization is an optimization method constructing a probabilistic model in order to evaluate parameter selections. When sets of parameters are selected, these are compared and the results are observed. The observed results are used as a measure of generalization of the model performance with the selected hyperparameters. The method then attempts to optimize the selection of parameters in order to improve the model.

The Bayesian optimization method thusly reduces the search space of hyperparameters using information available from previous evaluations of the model Snoek et al. [2012].

2.1.9 Artificial Intelligence*

Artificial Intelligence (AI) is an umbrella term that encapsulates all algorithms that show some intelligent behavior.

Machine Learning (ML), a subset of AI [Figure 2.7], is a system that learns on its own through experiences. A programmer does not explicitly program it. A ML process makes observations from data to identify possible patterns that can inform future decisions. Machine Learning aims to allow a system to learn by itself through experience without human intervention. Under the hood of many machine learning models are just plain statistics [Figure 2.8]. The major difference between machine learning and statistics is their purpose. Statistics is the mathematical study of data. Machine learning models are designed to make the most accurate predictions.

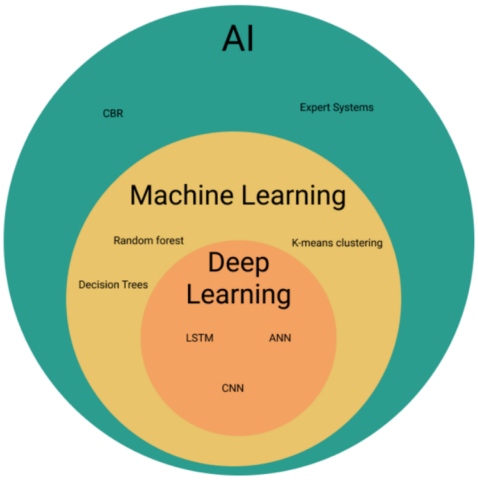


Figure 2.7: Venn diagram of Artificial Intelligense, Machine Learning, and Deep Learning

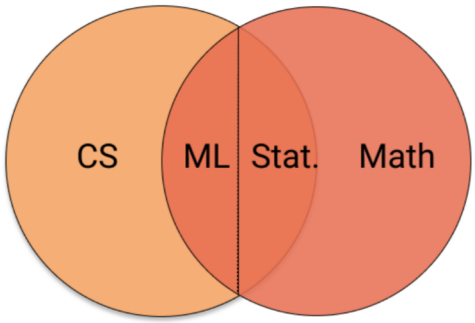


Figure 2.8: Venn diagram of Computer Science, Machine Learning, Statistics, and Mathematics

2.1.10 Deep learning*

Deep learning is a field within artificial intelligence which are inspired by how the human brain works. These neural networks are trained and used to accomplish tasks such as classification, clustering, natural language processing, predictive tasks, and much more.

Supervised learning

Supervised learning is a sub-field of machine learning where the focus is on training machine-learning methods through the use of labeled data. The method will attempt to access the data in correlation with the connected data label. Classical problems within supervised learning are classification problems where the prediction of a label is the desired result.

Unsupervised learning

Unsupervised learning focuses on creating machine learning methods using data without a label. The data has no "correct" label associated with it. Applications of unsupervised learning might be clustering, anomaly detection recommendation systems.

2.1.11 Convolutional Neural Network*

A **Convolutional Neural Network (CNN)** is a neural network architecture built using convolutional layers in order to extract information. Unlike fully connected neural networks, convolutional layers interpret data using perceptive

fields. These perceptive fields evaluate only sections of the input at a time until the whole input is processed. The convolutional layers attempt to extract features from the input data. The first layer extracts low-level features, while the next layer extracts higher-level features, and so on [Géron, 2017, p. 443-446]. Convolutional feature extraction is illustrated in Figure 2.9.

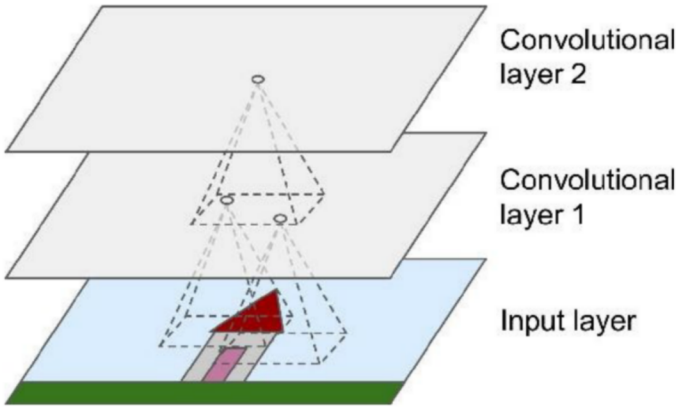


Figure 2.9: Figure of CNN layers from [Géron, 2017, p. 444].

Multiple kernels, or filters, are used to extract features from the data. The result of applying these filters is known as the feature map or the extracted data features. These feature maps extract lower and higher lever features from the original data by extracting spatial features, retaining the spatial relationship within the data. Such spatial features could be the curvature of a dog's ears in an image or the correlation of timestep data in a time series. As a result, convolutional networks have several applications within image classification, image recognition, natural language processing, and time series analysis.

2.1.12 Recurrent neural networks*

A **Recurrent Neural Network (RNN)**is an artificial neural network architecture that can work with data sequences of arbitrary length. Unlike feed-forward networks, RNNs consider the input data in conjunction with state information from a previous timestep. To accomplish this, the network uses feedback connections. The feedback connections serve state information from the previous time step to the intended node. This connection works as a short-term memory for the recurrent layers, saving information from the previous time step memory cells.

In the basic RNN, these memory cells retain minimal information, saving data only from the previous instance. As the RNN memory cell is defined by

the newest data introduced to the cell, previous information is encoded only in its effect on that data. Due to this, information is not stored for long in these memory cells, only retaining short-term memory data. The RNN memory cell is illustrated in Figure 2.10.

The RNN is able to process data of arbitrary length, meaning that it is well suited for natural language processing, time series analysis, and similar problems. In addition, the memory retained in the RNN makes it well suited to extract temporal relations in the data.

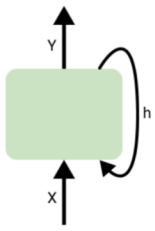


Figure 2.10: Figure of RNN feedback-cell.

In order to improve the performance of the RNN, new models have been created to address some of its shortcomings. One such model is the Long-Short Term Memory model (LSTM) [Géron, 2017, p. 469-472].

2.1.13 Long-Short Term Memory

Long-Short Term Memory (LSTM) is a type of recurrent neural network addressing some of the shortcomings of the RNN model, such as the vanishing gradients problem. The LSTM introduces a new memory cell, adding Long-term memory to the network.

The LSTM memory cells are comprised of two vectors, one for long-term and one for short-term memory, as well as an input gate (Equation (2.13)), output gate (Equation (2.14)), and a forget gate (Equation (2.12)). The forget gate allows for the memory cell to remove unneeded parts of the memory in order to replace it with new data from the input gate. The long-term memory retains some of its information while replacing other parts. A illustration of the LSTM cell is shown in Figure 2.11.

The long-term memory of the LSTM enables it to solve the RNN problem of vanishing gradients. The long-term memory enables the LSTM to store data at arbitrary intervals, as well as to detect long-term dependencies in the data.

The LSTM, like the RNN, is well suited for working on a series of data. The LSTM can analyze and predict long-term relations in a series of data, making it well suited for applications such as time series prediction or anomaly detection, natural language processing, and more [Géron, 2017, p. 492-493].

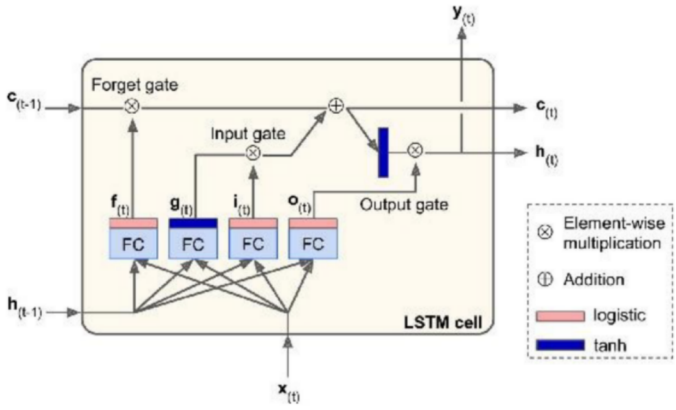


Figure 2.11: Figure of LSTM memory cell from [Géron, 2017, p. 492].

$$f_t = \sigma(X_t * U_f, +H_{t-t} * W_f) \tag{2.12}$$

$$i_t = \sigma(X_t * U_i, +H_{t-t} * W_i) \tag{2.13}$$

$$i_t = \sigma(X_t * U_o, +H_{t-t} * W_o) \tag{2.14}$$

- X_t: input to the current timestamp.
- U_f: weight associated with the input.
- H_{t-1}: the hidden state of the previous timestamp.
- W_f: weight matrix associated with hidden state.

Stateful and stateless LSTMs

A LSTM takes three vectors as input. The previous hidden state H_{t-1} which is the short term vector described above, the previous cell state C_{t-1} which is the long-term vector, and the input vector X_t. These are not to be confused with the cells internal weights W_i which is the weight matrix for the hidden state vector, and U_i, the weight matrix for the input X_t.

During training, these weight matrixes are shaped by backpropagation. The hidden state and internal state vectors are not touched by backpropagation but will constantly be changing during forward passes through the network.

A stateful LSTM refers to an architecture where the hidden state and cell state are initialized once but never reset during training. This is typically when the batches in the training set are dependent on each other, like one long time series.

In a stateless LSTM the hidden states are reset many times during training, typically at the end of each batch. A typical example is if the training data is a set of unrelated sentences.

2.1.14 Autoencoder*

Autoencoders are neural network models used to learn efficient representations and encodings of data. Through an unsupervised learning process, autoencoders aim to reduce the dimensional complexity of data. The models store information in order to encode and decode input data, efficiently storing representations of the data used to encode and decode data. Due to the autoencoder's ability to encode and then reconstruct data, it is well suited for the dimensional reduction of data. Autoencoders consist of two parts; the encoder and the decoder. The encoder is tasked with reducing the data in order to create a coding representation from the data. This coding is then used by the decoder in order to attempt to reconstruct the input data. An illustration of an example architecture for an autoencoder is shown in Figure 2.12.

Through feature mapping, the encoder becomes an efficient feature detector and extractor. Due to the reduced dimensionality of the coding, the encoder becomes sufficient at reducing the noise within the data, extracting the most important features. As a future extractor, autoencoders are well suited for pre-training neural networks, extracting essential features. At the same time, autoencoders can become successful generative models. The decoder is proficient at reconstructing input data from the coding layer representation, meaning it can also generate new data. The reconstructive abilities of the decoder enable the generative model to create new data, similar to the training data used when creating the model [Géron, 2017, p. 506-508].

2.1.15 Day of the week formula

The book Hale-Evans [1998] has an algorithm which can be used to calculate the day of the week from a date, for example 14.01.1996.

$$day\_of\_the\_week = (YC + MC + CC + DC - LYC) \mod 7 \tag{2.15}$$

- YC = Year code.
- MC = Month code.

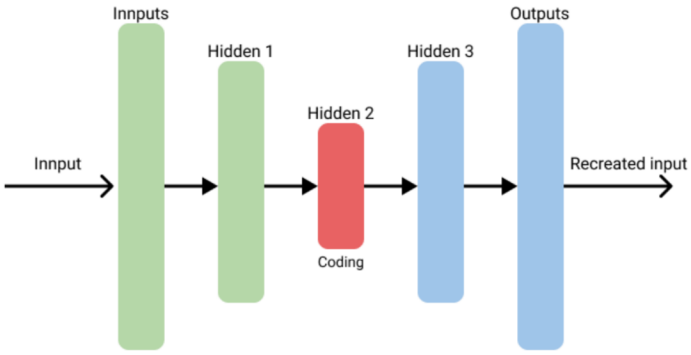


Figure 2.12: Figure of a Stacked Autoencoder architecture with one hidden encoder layer, and one hidden decoder layer.

- CC = Century code.
- DC = Date number.
- LYC = Leap year code.

The year code is calculated using $YC = (YY + (YY)/4) \mod 7$, where YY is the last two digits of the year. For the year 1996, $YY = 96$.

The *Month Code* is derived from Table 2.1. The date number DC is just what day of the month it is.

The century code CC is 4. The leap year code LYC is 1 the month is january or february of a leap year.

2.1.16 Student's t-test

The *student's t-test* is a statistical significance test. It is a method of testing hypotheses about the mean of a small sample drawn from a normally distributed population when the population standard deviation is unknown. In other words, it is a method to check if the difference in mean between two groups, is because of chance.

The method formulates a null hypothesis, which states that there is no effective difference between the observed sample mean and the hypothesized population mean. In other words, it assumes that both groups are equal.

It uses Equation (2.16) to calculate the t-value.

Table 2.1: Month Code table

Month	Code
January	0
February	3
March	3
April	6
May	1
June	4
July	6
August	2
September	5
October	0
November	3
December	5

$$t = \frac{\sum D/N}{\sqrt{\frac{\sum D^2 - \frac{\sum D^2}{N}}{(N-1)(N)}}} \tag{2.16}$$

where D is the difference between each sample in the two groups , aka $x_i - y_i$. So $\sum D = \sum x_i - y_i$

We can find the p-value by looking it up in the t-table by using our degrees of freedom, which are $N - 1$. If the p-value is below our α of 0.05 then we can reject the null hypothesis that the groups are equal.

2.2 Structured Literature Review Protocol*

A structural literature review (SLR) was conducted to find related work for this project. The SLR process is based on Anders Kofod-Petersen [2018]. This section will briefly describe our process, deviations from the process, and changes made during the process.

2.2.1 Step 1: Idendification of research

In order to retrieve all relevant literature on the topic at hand, we defined our Research Questions (RQ) and Research Terms (RT).

We included the best-known search engines as our sources:

ACM digital library, IEEE Xplore, ISI web of knowledge, ScienceDirect, CiteSeer, SpringerLink and, Google Scholar

Our search terms are listen in Table 2.2 and Table 2.3. We used a Notion database to keep track of all the papers. The whole database is available here Sindre Sivertsen [2021d]. We used another Notion Database to keep track of our searches and track how we found each Research Paper Sindre Sivertsen [2021c]

Our method for adding literature to the Notion database was:

1. Pick a source search engine
2. Search the search engine applying AND $\wedge$ and OR $\vee$ using our search terms
3. Add the Search Term and Source combination to the Search Term Database.
4. Title and abstract inclusion screening. Add all remotely relevant papers to Notion, and link them with the correct Search Term Database.
5. Continue until relevant papers are far in between
6. Repeat the whole process with a new search engine

Devations

We did end up with some deviations to the algorithm above. Some papers we found by accident when researching the topic. They were deemed too important not to add. Their search terms were too general for us to consider adding to the search terms without including several other non-relevant papers. All papers found by accident were similarly linked with a search term table row documenting how the paper was discovered.

Some of the best papers we found through the structured review contained references to other relevant papers, some of which were of high enough quality to add to our list of sources. These were also added to Notion.

The term *Anomaly detection* was removed (11.10.2021) as a search term after our goals shifted away from anomaly detection and towards time series prediction. This change was documented in our *Decision Matrix*, which is publicly available Sindre Sivertsen [2021a].

Name	Group 1	Group 2	Group 3
Term 1	CNN	~~Anomaly detection~~	Autoencoder
Term 2	Convolution Neural Networks	~~Extreme values~~	Encoder Decoder networks
Term 3		~~Outliers~~	

Table 2.2: Search Terms table 1

Name	Group 4	Group 5	Group 6
Term 1	Time-series	LSTM	E-commerce
Term 2		Long short term memory	Sales
Term 3			

Table 2.3: Search Terms table 2

2.2.2 Filtering by Title and Abstract

Working with six different sources and using a manual but an orderly template for processing papers, we decided to do the rough filtering while searching for papers in order to avoid adding a vast amount of irrelevant papers to our Notion database, thus saving time.

In order to filter out papers, we had three Inclusion Criteria (IC).

IC1 The study's main concern is time series forecasting.

IC2 The study's should not be older than from 2015.

IC3 The study focus on CNN, autoencoder, or LSTM. Or a statistical method.

IC1 excludes papers with a topic not relevant to our goals, such as time series classification and anomaly detection. IC2 was then added to reduce the number of relevant papers, thus reducing the scope of the structured literature review. Through initial research on the topic of time series prediction, we found few relevant papers from before 2015. Newer research was usually more relevant due to the more modern approaches applied, and thus this criterion was introduced. However, a small number of relevant papers published prior to 2015 were found to have relevance, and an exception was made for these papers. Lastly, IC3 was added to keep the papers within our defined scope of methods. 12.10.2021, we added "Or a statistical method." to IC3. This is documented in the decision matrix [Sindre Sivertsen, 2021a].

Each Inclusion criteria add a maximum of 1 point for each paper. For IC3, if the paper just covered one of the four listed methods, it would get 0.25 points. If a paper scored 0 points on either of the Inclusion Criteria, it would be filtered out.

2.2.3 Quality Assessment

In the final stage of filtering, the goal is to assess the quality of the work. This was done using a set of quality criteria for further screening.

- QC 1 The study abstract is concise and describes the aim and results of the study.
- QC 2 The publisher of the paper/study is a reputable scientific source.
- QC 3 Is there a clear statement of the aim of the research?
- QC 4 Is the study put into the context of other studies and research?
- QC 5 Are system or algorithmic design decisions justified?
- QC 6 Is the test data set reproducible?
- QC 8 Is the experimental procedure thoroughly explained and reproducible?
- QC 9 Is it clearly stated in the study which other algorithms the study's algorithms have been compared with?
- QC 10 Are the performance metrics used in the study explained and justified?
- QC 11 Are the test results thoroughly analyzed?
- QC 12 Does the test evidence support the findings presented?

The criteria are used with a scoring system. Yes (1 point), to some degree (0.5 points), and no (0 points). The maximum possible sum from the Quality Assessment is 12. Adding the points from Inclusion Criteria increases the maximum possible sum to 15.

The quality assessment resulted in a score range of [7.33, 15]. All papers below "7" were filtered out before the Quality Assessment. The overall list is public Sindre Sivertsen [2021e]. In order to limit the scope of papers and ensure the quality of the selected, a threshold of minimum 10 points was set for a paper to be included.

The final list of included papers contains 23 papers. A screenshot of where we drew the line for papers to include is presented i the appendix section 8.9.

2.3 Motivation*

This section will cover the underlying motivation for the inclusion of the different methods and theories in the literature review.

2.3.1 E-commerce and Sales predictions

With the proposed goal and research questions, the problem presented in this thesis is not to predict sales in an E-commerce setting. Despite this, the argument is made that the type of data attained from user interest on products should be within a similar data distribution and thus a similar problem space. Considering this, it is necessary to include a literature review of current solutions in sales predictions and an E-commerce setting. Researching the current state-of-the-art methods and theory creates a context of what methods have already been investigated and where improvements can be made.

2.3.2 Deep learning methods

Through the research of the current state-of-the-art forecasting methods, it is clear that deep learning frameworks are currently at the forefront of time series forecasting. Although statistical methods have long been state-of-the-art, current research suggests that deep learning methods can be used to improve predictive accuracy and reduce predictive error. Research from Makridakis et al. [2018] comparing statistical methods and deep learning methods for time series forecasting suggests that deep learning methods are superior if there is enough data to process. Papers such as Laptev et al. [2017] suggests that methods such as Autoencers and LSTM are well suited for time series predictions. Further research into these and other models was therefore needed in order to assess the current development of time series prediction frameworks.

2.3.3 Hybrid models

Through initial research in the time series prediction domain, we discovered a new framework for making time series predictions. The paper from Zhao et al. [2019] covers the use of a deep learning framework using a convolutional autoencoder coupled with a LSTM in order to achieve more accurate predictions. The proposed framework showed great results on data with high fluctuation, which would warrant the investigation of its use on our problem space and data. This hybrid method showed greater predictive ability than the individual parts could achieve independently. With this, we made a case to find other hybrid models or connected models in order to verify the predictive superiority of hybrid models. This led us to other papers such as Khan et al. [2020] and Bowen et al. [2020].

Chapter 3

Related work

In order to find a fitting solution to the goal proposed in Section 1.3 and its related research questions, we need to assess what advances have been accomplished within the problem space. We assess state-of-the-art methods in time series prediction and loss functions. With this, we are able to compare current solutions and methodology in order to answer our research questions.

This chapter starts with Section 3.1 introducing current and previous advances within E-commerce time series forecasting. Section 3.2 explores the use of deep learning method in comparison with statistical methods. Later, Section 3.3 introduces the use of global and local methods, as well as univariate and multivariate methods. Section 3.4 introduces current state-of-the-art hybrid methods for forecasting. Multi-step ahead predictions are pressented in Section 3.5, before a critique of LSTM usage in literature is given in Section 3.6.

Lastly, selected literature from the *Related work* section, which creates the basis for further experimentation and methods, are presented in Section 3.7.

3.1 Forecasting E-commerce*

User click rate on consumer E-commerce goods is a narrow domain, and we are unable to find any research that has yet been done within this particular field. However, one can argue that user interest in consumer goods follows the same distribution as online retail sales. Both domains will have the same seasonality with weekly and yearly periodic fluctuations. Additionally, the domains contain the same kind of products and categories. Both domains are dictated by the same external factors, such as media, commercials, and trends. Forecasting retail sales should translate well to E-commerce click rate.

Ramos et al. [2015] did a comparative study on exponential smoothing methods based on state-space models such as ETS and ARIMA. The aim was to measure the forecasting performance of the two modeling frameworks when applied to retail sales data of five different categories, containing univariate time series regarding sales of women's footwear. The domain proved difficult due to strong trends and seasonal patterns.

The models forecasting ability were evaluated through several loss functions, such as RMSE, MAE, and MAPE. Both ETS and ARIMA achieved similar results with both one-step and multi-step forecasting. Their results show that multi-step forecasts are generally more accurate. This is argued to be a result of multi-step to incorporate more up-to-date information. Neither of the models is best suited for all circumstances.

ETS and AIRMA have also been shown to be effective in other instances. Chu and Zhang [2003] shows that Holt-Winters exponential smoothing and ARIMA perform well when macro-economic conditions are relatively stable. However, when economic conditions are volatile, ANNs outperform linear methods. Despite this, the existence of stable data is not guaranteed. Often, the data contain nonlinear characteristics, increasing the complexity of the problem. Weng et al. [2020] writes that ARIMA, SARIMA, and Holt-Winter's Exponential Smoothing model etc., is inconsistent with the actual changes in the sales of retail stores. Results are usually unstable due to their in-applicability in the processing of nonlinear relationships. Despite this, statistical methods have the advantage of high interpretability, whereas more complex machine learning methods are limited by the high complexity and reduced interpretability.

In comparison to Weng et al. [2020], the paper Bowen et al. [2020] wanted to use machine learning to maintain the tradeoff between interpretability and consistency with actual changes in retail sales. Using a hybrid ARIMA and Back-propagation method, increased forecasting metrics was achieved through giving up the interpretability of the pure statistical methods. The same gains in forecasting ability but loss of interpretability was found by Zunic et al. [2020]. They used pure machine learning methods such as the LightGBM, an ensemble learning method based on the XGB model, and the Prophet model, with good results.

Bowen et al. [2020] finds that traditional models such as ARIMA have trouble forecasting E-commerce due to the fast-moving domain of consumer retail buying patterns. Additionally, their connection to external factors, such as holidays and price changes, increases the complexity further. The machine learning models performed better, with lower predictive error and good interpretability. However, the LightGDM model outperformed the Prophet model because of the models limitation to univariate time series.

Further experimentation with machine learning methods is done by Bandara

et al. [2019], through the use of deep learning methods. An accurate E-commerce sales forecasting model is generated using LSTM Neural Network. Similar time series are pooled into clusters where univariate and global models are trained on each cluster. The aim was to improve E-commerce sales forecasting by exploiting sales correlations and relationships available in an E-commerce hierarchy.

Data preprocessing is done using a forward-filling strategy to input missing sales observations in the dataset. The data was then normalized to accommodate the high sales volume ranges. They then used the mean of the sales of a product as the scaling factor.

Bandara et al. [2019] applied two different techniques for grouping data. The first approach was based on domain knowledge. Sales ranking and the percentage of zero sales were used as the primary business metrics to form groups of products. Group 1 represents their most popular products (67% of sales), group 2 represents their more unpopular products (33% of sales), and group 3 represents the rest.

The second approach was based on time series clustering. K-means clustering was used on a set of time series features to identify groupings. The first two features were business-specific features, namely "sales.quantile" and "zero.sales.percentage". The rest of the features were time series specific. A silhouette analysis was utilized to determine the optimal number of clusters in the K-means algorithm.

The results outperformed the state-of-the-art univariate forecasting techniques. Thereby concluding that E-commerce product hierarchies contain various cross-product demand patterns and correlations. Exploiting these relationships are necessary to improve the sales forecasting accuracy in this domain.

With the effort to advance the predictive capabilities of models in E-commerce, previous work done within the problem space is highly relevant as a starting point. This shows what has already been accomplished and what new methods have not yet been attempted.

3.2 Statistical methods versus Neural Nets*

Statistical methods like ARIMA and exponential smoothing were once the state-of-the-art methods for forecasting time series. In recent years Deep Neural Networks have attracted more attention. Makridakis et al. [2018] wanted to objectively test these up-and-coming machine learning models to compare with old state-of-the-art methods. Performance was evaluated across multiple forecasting horizons using a large subset of 1045 monthly time series. Among the evaluated machine learning methods are: *Bayesian Neural Network (BNN)*, *Multi-Layer Perceptron (MLP)*, *K-Nearest Neighbor regression (KNN)*, *Recurrent Neural Network (RNN)*, and *Long Short Term Memory neural network (LSTM)*

Among the statistical methods evaluated are: *Naive2*, *ETS*, and *ARIMA*. They conclude that the statistical methods outperformed all ML methods in terms of accuracy, measured for both simple one-step-ahead and multiple steps ahead methods.

The paper Makridakis et al. [2018] highlights some of the drawbacks of ML methods. Especially their complexity, their computational cost, their lack of explainability and their inability to show certainty in their predictions. However, the study has gotten some well-defined criticism. The authors Cerqueira et al. points out that Makridakis et al. [2018] does their experiments on datasets of too-small sample size. Their largest time series sample size among the 1045 datasets contained only 144 data points, while the smallest contained 118. They hypothesize that these datasets are too small for an ML method to generalize correctly.

Cerqueira et al. [2019] therefor conduct a similar study on 90 univariate time series, in which all the datasets have a sample size above 1000. Statistical methods are evaluated against machine learning methods at different sample sizes, in order to test whether the sample size has an impact on performance. They conclude that sample size impacts machine learning methods drastically. Statistical methods outperformed ML methods up to around a sample size of 130 data points. However, in tests conducted with larger sample sizes, the machine learning methods were generally shown to outperform the statistical methods.

From the discussion originating with the two papers described above, it should be clear that machine learning methods are to be preferred in the event of sufficient data. Hewamalage et al. [2021] confirms this statement. They further conclude that even with the higher computational costs associated with ANNs, such computational costs are feasible because of the general availability of cloud computing. Complex deep learning methods now have benefits over simpler statistical methods in many forecasting cases. Although such statistical methods are reliable and produce a relevant explainable predictive baseline, modern deep learning methods can accomplish higher predictive accuracy.

The papers presented above show the relevance of exploring new and more complex methods to improve time series prediction. Deep learning methods such as LSTMs have high predictive accuracy, and exploring these methods further may prove to be advantageous. Nevertheless, statistical methods supply a well-defined predictive baseline for a time series. Additionally, statistical methods have the advantage of improved explainability, making these methods more readily understandable. With this in mind, using well-defined statistical methods as a baseline prediction to evaluate new predictive methods should be considered. Only comparing highly complex deep learning methods may prove difficult if they cannot make accurate predictions. A baseline should therefore be a meaningful way of evaluating more complex methods.

3.3 Model Structure*

The structure of the forecasting model is an important aspect when the goal is to forecast multiple time series. A simple approach would be to make one model for each time series. One could assume that each series are independent, but this assumption will probably not hold water for all domains. For example, in the E-commerce domain, people who buy shaving cream probably also buy a razer, as shaving cream will not have much use of its own. An alternative is to build a more complex model that looks at a bigger picture. This section will explore different state-of-the-art model structure approaches.

3.3.1 Global versus Local methods

On the topic of having to forecast many time series as a group, the paper from Montero-Manso and Hyndman [2021] provides a good overview. The article points to two significant disadvantages for univariate models on a cluster of series. The number one shortcoming is the sample size. The second is scalability. Scalability is a problem when a group of time series each requires a separate model that requires human intervention. Forecasting a cluster of time series in this manner is called *the local approach.*

A univariate alternative to a local approach is *the global approach* [Rabanser et al., 2020]. The global approach works by pooling all series data together, fitting a single univariate forecasting function. It prevents over-fitting because of the larger sample size. The global method has been introduced to exploit the natural scenario where all series in the set are similar or related. An example given by the authors is the demand for fictional books follows a similar pattern for all subgenres, stores, or cover designs. The idea behind this is the strong assumption that all the time series in the set come from the same process.

This exact method was used by Bandara et al. [2017]. Their domain has similarities to ours. They want to forecast a database of E-commerce time series from Walmart.com. They argue that when building global models for a time series database, the models are potentially trained across disparate series, which may be detrimental to the overall accuracy. They suggest building separate models for subgroups of time series. These groups can be selected based on domain knowledge, which proved to be the best option. With the absence of domain knowledge, they propose an automatic grouping mechanism to cluster series together. Their method achieves consistent improvements over the baseline LSTM model. And conclude that exploiting similarities of multiple time series in one model is a competitive method.

Montero-Manso and Hyndman [2021] show that even if the strong assumption that the same process generates all the time series analyzed by a global the

method is false; the global method will pay off in forecasting accuracy. The paper argues that global and local methods for forecasting sets of time series are equally general. The global method is neither restrictive nor requires similarity or relatedness in the set. But they point out that generalization of global models assumes groups of independent time series. Under heavy dependence, global models lose their beneficial performance guarantees.

The paper Hewamalage et al. [2021] comes to the same conclusion. Stating that even on datasets that involve many heterogeneous series, the strong modeling capabilities of RNNs can drive them to perform competitively in forecasting accuracy.

3.3.2 Univariate or Multivariate time series

Bandara et al. [2017] points out that statistical methods, like ARIMA, are bound to univariate time series. In the world of Big Data and lots of time series that correlate with each other, treating each time series separately and forecasting each in isolation could isolate information regarding big picture trends. Bandara et al. [2017] argues that the ability to make models that can be trained globally across all series holds a competitive advantage over models like ARIMA and ETS. Such a model would simultaneously use multiple time series as input to predict future values for the time series.

The paper Rabanser et al. [2020] has some good arguments when comparing univariate versus multivariate models. Both multivariate and global univariate methods work on groups of time series. However, global methods have the advantage of being more applicable because it does not require observations of multiple time series at the time of forecasting. Also, multivariate time series models work on groups that are supposed to have some form of dependence between them, while global models work on any group. When such a dependency exists, global method will not capture it directly, unlike multivariate models. Hewamalage et al. [2021] states in their "*7. Future directions*" chapter that complex forecasting scenarios, such as a retail sales forecast, the sales of different products may be interdependent. Forecasting in such a context could require a multivariate model.

Laptev et al. [2017] wanted to make a single time series model to accurately make time series predictions during special events. Extreme event prediction depends on numerous external factors, including weather, city population growth, or marketing changes. Laptev et al. [2017] propose a global, multivariate, autoencoder, LSTM network. Their results are promising. They outperform their existing proprietary model by up to 18% on Uber data. They also show the model's generalization power by training on Uber data but then testing it on the public M3 dataset, where they achieve an above-average result. In Laptev

et al. [2017] discussion, they point out three criteria for choosing a neural network over a statistical method: (a) number of time series to the model is high, (b) Length of the times series are high, and (c) correlation among the time series. Our problem domain meets all these criteria. The third (c) criteria are worth mentioning because it directly contradicts the argument proposed by Montero-Manso and Hyndman [2021]which states that in a set of interdependent time series, a global model lose their beneficial performance guarantees, because a global model assumes independence. This again contradicts Hewamalage et al. [2021], which explicitly states retail forecasts, which, as argued, is similar to E-commerce, as a perfect example for a global model. Of all the papers we found, none of them concluded that a global method on time series that have some form of interdependency would directly hurt the model.

Sen et al. [2019] proposes a method to handle predictions of thousands of interdependent correlating time series. Their proposed method is largely based on two components. The first component is based on the work of Yu et al. [2016], which proposes a Temporal regularized matrix factorization (TRM) for high-dimensional time series prediction. The idea is that the TRM can look at all the time series and capture the global patterns during prediction. The TRM can supposedly handle as much as 50.000 time series. The output of the global model is used as a covariate to a final, local temporal convolution network. This final model will then focus on local per time series properties, as well as properties from the global dataset.

3.4 Hybrid frameworks for Time-series forecasting*

Improvements in time series forecasting have emerged in the last few years with the introduction of deep learning. Previous state-of-the-art methods, such as the ARIMA model, have been exchanged for deep learning methods using convolution and recurrent ANNs to improve accuracy. One such improvement strategy has been to introduce hybrid models. These models are comprised of different models with unique abilities, helping to increase the accuracy of the connected model.

One example of such a hybrid model is the ARIMA-BP model explored in Bowen et al. [2020]. This thesis explores the predictive abilities of the ARIMA model and a Back-Propagation Neural network on a time series problem of sales forecasting. The ARIMA and BP models were applied to the problem, with the ARIMA outperforming the BP model. However, by combining the two models, the hybrid model was able to achieve higher predictive accuracy than either of the two models accomplished individually.

As we have explored in Section 3.2, the introduction of more advanced neural

networks often achieves better predictive results than the old ARIMA model. We, therefore, argue that it should make sense to look further into other hybrid models, exploring these new and improved models.

The hybrid combination of different ANNs is explored by Khan et al. [2020], which applies a CNN combined with a LSTM-Autoencoder. This method is used to explore the predictive ability of the hybrid model on electricity forecasting in residential and commercial buildings. The findings of the thesis conclude with the same result as proposed above. The aforementioned method is a hybrid framework based on convolution, LSTM, and Autoencoder. Convolution is used to extract features from the input data, while the LSTM-Autoencoder extracts temporal dependencies between the sequences. The proposed method is applied to the electricity forecasting problem alongside other models such as ARMA, SVM, SVR, and others. The hybrid framework outperformed the other models at prediction using multiple performance metrics such as MSE, MAE, RMSE, and MAPE, as well as other hybrid deep learning methods such as a CNN-LSTM.

Similar to the framework described above, another framework applying a combination of the same methods was defined in Zhao et al. [2019]. This model differs through architecture selection, creating a Convolutional Autoencoder instead of a LSTM Autoencoder. The Convolutional Autoencoder is used to extract and reduce dimensional features before the LSTM extracts the temporal features. Similar to the previously mentioned method, this "CNN-AE-LSTM" hybrid method is tested against several other current predictive models. The proposed method outperforms methods such as LSTM, ARIMA, and SVR, achieving a much lower predictive error than the compared methods.

Other attempts to improve predictive forecasting have also shown promising results. Frameworks such as with the proposed AE-LSTM from Van Hoa et al. [2021] have shown the use of an autoencoder to increase the accuracy of the predictions over a basic LSTM method. This thesis focuses on foreign exchange rate forecasting and shows the increased accuracy achieved using an Autoencoder and LSTM over a basic CNN or LSTM network. Similarly, Zhang and You [2020] uses a Gated Dialated Causal Convolutional Encoder-Decoder in network traffic forecasting. It shows the decrease in predictive error with an advanced Convolutional Autoencoder compared to a more simple deep learning method like a LSTM.

The literature seems to favor a more complex hybrid ANN over a simpler network.

3.5 Multi-step ahead*

Intending to forecast future values, there is a question of how far into the feature a meaningful prediction can be given. As stated in Section 2.1.3, there are two

ways to achieve this. Either through the use of multiple single-step predictions or a multi-step-ahead forecast.

Hewamalage et al. [2021] presents a section regarding multi-step-ahead forecasting. Citing Ben Taieb et al. [2011] works and their findings that using a multip-input multi-output (MIMO) strategy is advantageous over the recursive single-step-ahead. This is because the MIMO strategy incorporates the interdependencies between each time step rather than forecasting each time step in isolation. They also found that the MIMO strategy voids error accumulation over the prediction time steps. Ramos et al. [2015] findings also support the MIMO strategy over the recursive single-step-ahead strategy.

Hewamalage et al. [2021] suggests that the best input-to-output size ratio is $window\_input\_size = 1.25 * output\_window\_size$. With this in mind, the use of multi-step-ahead prediction appears to be the most appropriate approach to use.

3.6 Criticue

The book Bharadi and Alegavi [2021] describes the decoupling of resetting hidden states during training of LSTM, and describes the issue of needing a fixed batch size. We found countless internet forum posts and some articles that touched on this issue, a few good solutions to the problem were found.

The literature seems to give few answers for how they handle hidden state resets, if they are using stateful or stateless, and if stateful, and how they handle the fixed batch size problem.

Bandara et al. [2017] has the following sentences

> "All training patches relevant to a particular time series are read as one sequence. Therefore, the LSTM state needs to be initialized for each series".

This indicates that they are using a stateful LSTM [Section 2.1.13]. They are training global models across multiple time series, but they do not describe how or when they are resetting the hidden states. If they pass through historic relevant data before predicting the test set, or how they handle the equal batch size constraint.

Hewamalage et al. [2021] describes in great detail the different architectures used, but they do not explicitly reveal how they handle initializing and resetting of hidden states.

We found one paper that explicitly names resetting hidden states as a tool Smyl [2020], but no explanation of how this was done.

3.7 Selected Literature

This chapter has introduced multiple theoretical sources of information presenting the theoretical influences for this master's thesis.

The paper by Zhao et al. [2019] serves as the inspiration for the use of the CNN-AE-LSTM model in this thesis. Influenced by Zhao et al. [2019] and Cerqueira et al. [2019] the ARIMA/SARIMA model, along with the local univariate LSTM model is used as baseline models in this paper. As we have seen from Cerqueira et al. [2019], the use of neural network models is preferred when the amount of data is sufficient, and we can therefore assume the use of new neural network models is preferred.

Additionaly, Montero-Manso and Hyndman [2021], Bandara et al. [2017], Laptev et al. [2017] inspire exploration of new model structures for LSTM and CNN-AE-LSTM models. Montero-Manso and Hyndman [2021] and Bandara et al. [2017] discuss improvements achieved through use of global models, while Laptev et al. [2017] suggests the use of multi-feature multivariate models for improved performance.

For use with such model structures, Bandara et al. [2017] also suggests the use of groupings of data, which in turn serve as inspiration for grouping of time series into datasets for experimentation.

These papers are some of the papers discussed, which serve as inspiration for the research conducted in this master's thesis.

Chapter 4

Architecture/Model

In this section, we propose a framework architecture intended to solve the goal presented in Section 1.3. The model composition and selection of tuning methods are presented for each of the models that are used. Section 4.1 starts with presenting the SARIMA model, the initial predictive baseline, before the LSTM model is presented in Section 4.2. Lastly, Section 4.3 presents the model design of the convolutional autoencoder used in the convolutional autoencoder and LSTM model.

4.1 SARIMA baseline

As a means to measure the predictive ability of new models, a comparison with current well-established methods can be useful. Two such predictive methods are the ARIMA and SARIMA models.

As described in Chapter 5 the available data is both non-stationary, in addition to having a varrying degree of seasonality. Due to this seasonal component, a SARIMA model would be the best fitting one of the two.

4.1.1 Model selection

In order to select useful SARIMA models, tuning methods were introduced. With the aim of finding a set of optimal model parameters, two different methods of model tuning were used. Both Grid Search and Auto-ARIMA is implemented as a means of tuning the models.

As the models are aiming at making a 7 day prediction, Grid search is conducted using a 7 day validation set as a measurement for accuracy. Each parameter set is used to make a 7 day prediction, where this prediction is compared to

the true values in the validation set. An error metrics such as the MAE, MSE, MASE or others can be used to find the best model. When a set of parameters are selected, the validation data is added to the training set, and the model is retrained before it attempts to make a new prediction on a test set. This is then used as a measure for the model.

Additionally, Auto-Arima is used as a second method for tuning the ARIMA and SARIMA models. This approach uses Bayesian optimization for parameter tuning, attempting to find the best hyper-parameters for the models. Using an error metric such as the MSE, the auto-arima model is tuned to find the best parameters for each time series.

However, although the initial intention was to use both auto arima and Grid search, a decision was made to reduce the number of experiments. The auto arima framework was discovered to make a tuning selection that was on-par or better than the grid search models. This is likely due to the fact that the grid search models became so specialized that they overfit the training data to such an extent that the auto-arima perform much better. Thus, as the ARIMA and SARIMA models only serve as a benchmark, only the auto-arima framework will be used during tuning.

Lastly, due to the highly seasonal component in the datasets used in this project, the SARIMA model is preferred due to its ability for seasonal prediction.

4.2 LSTM Baseline

The LSTM model is described in Section 2.1.13, and is the current state-of-the-art method for making time series predictions. Comparing new models against the current state-of-the-art models serves as a good measure of whether or not the new models are useful.

4.2.1 Global and local methods

Unlike the SARIMA model serving as the initial baseline, the LSTM model is able to work both as a local and a global methods.

First, a local model is created. This can be directly compared against the SARIMA model described in the previous section. This model only uses one time series as data for training, validation and testing, similar to the SARIMA model.

However, the LSTM model differs vastly from the SARIMA model in that it is able to be used as a global model. Unlike the local method using only one time series for training and testing, a global model is able to use multiple time series. Using a set of time series both for training, validation and testing, the LSTM should be able to increase the predictive ability and accuracy by increasing the

amount of data available. Additionally, the use of a global model enables the LSTM to extract features from different time series that hopefully can be used to make more accurate predictions on the original dataset.

4.2.2 Univariate and Multivariate

The LSTM model is configurable as both a univariate and a multivariate model. While the univariate model only uses one input variable per time-step, a multivariate model takes multiple inputs for each time-step. For seasonal data, one could then extract information from the dataset, adding multiple new values per time-step in order to encode information such as season or month.

4.2.3 Model selection

In order to find a fitting LSTM model, hyperparameters are tuned. With the use of the *optuna framework* we are able to tune the hyperparameters of our models. The *optuna* framework applies Bayesian optimization as described in Section 2.1.8.

This approach of hyperparameter tuning is used with each of the LSTM versions. The tuning of the univariate and multivariate and local and global are all done through the use of Bayesian optimization using optuna.

4.3 Hybrid Model Architecture

We propose a prediction framework based on the one presented in Zhao et al. [2019]. Creating a Convolutional autoencoder and LSTM method for time series prediction should provide some benefits. Primarily, the combination of methods should help with increasing the prediction accuracy in data with high fluctuations. Unlike the univariate method used in Zhao et al. [2019], we intend to improve upon the method by experimenting with different model structures to capture interdependent relationships in a time series cluster. Such a method has never been explored in the E-commerce forecasting domain, warranting the exploration of the method's applicability to the problem space.

The proposed framework is comprised of two parts: the convolutional autoencoder and the LSTM.

The convolutional autoencoder process the input data, extracting a feature set by deconstructing the data. This is done through the encoder. After this, the decoder is used to reconstruct the input data. By doing this, the noise in the input data should be decreased to some extent.

The second part of the architecture is the LSTM module. This module is intended to extract the temporal features of the dataset to predict future values in the time series.

An illustration of the described model is shown in Figure 4.1.

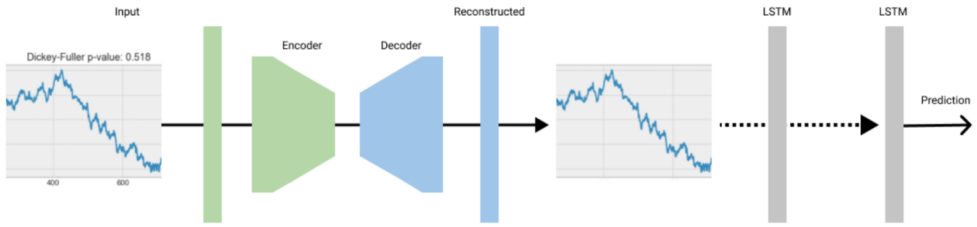


Figure 4.1: Illustration of a CNN-AE + LSTM network.

Finally, the complete framework proposed in this thesis connects these two models, creating a convolutional autoencoder and LSTM framework for making predictions. These predictions are intended to function on time series data with high fluctuations, making accurate and less error-prone predictions than simple statistical or deep learning methods.

As a means to validate the proposed method, predictive benchmarks should be provided. Established methods such as the SARIMA method and the LSTM method are well suited for this. These methods create a baseline that can be used to establish the comparative performance with the proposed method.

The motivation behind the proposed framework is explored in further detail later in Chapter 8.

4.3.1 The Auto-encoder

The first part of the proposed hybrid model is the convolutional autoencoder used to process the input data. As described, the task of the autoencoder is to encode and reconstruct the input data, while decreasing the noise.

The design of the autoencoder is based on the thesis by Zhao et al. [2019], detailing a convolutional autoencoder comprised only of 1 dimensional convolutional and trans-convolutional layers.

In order to find a well-suited autoencoder model for our problem-space, we designed a model to reconstruct the input data with high accuracy, while still removing noise from the dataset. Due to the fact that all the data in the different datasets share similar characteristics a shared autoencoder model is developed.

The autoencoder consists of convolutional and trans-convolutional layers. There are however other layers and methods that are not used in the autoen-

coder design. Methods such as batch-normalization and MaxPooling are not used. This is because adding these measures results in quite poor reconstructive results, likely due to the high volatility of the data. Other layers, such as dense, fully connected layers, which often represent the coding in the autoencoder, is not used either. This is because it was found to be an unneeded increase in the complexity of the model, as the convolutional layers were well suited to solve the task on their own.

There are two reasons for using a shared model design. Primarily this is due to the fact that during experimentation with the autoencoder models architecture, it was found that the same autoencoder performed similarly on each of the individual time series. Secondly, this was a decision that was made in connection with the limited time available for experimentation and optimization of models.

4.3.2 LSTM

The second part of the hybrid model is the LSTM model. This model is attached at the end of the autoencoder, using the reconstructed values from the autoencoder as input data.

Like the LSTM described in Section 4.2, global, local, multivariate, and univariate models are explored.

Chapter 5

Data

The dataset available for this thesis creates the basis for model selection and design. This section presents the available data for this project. Exploration of the dataset is presented in Section 5.1, and the selected time series contained in the selected datasets are presented in Section 5.2

5.1 Data Exploration*

In this section we will describe the main dataset given to us by Prisguiden.no. The full Jupyter notebook exploration is located at Sindre Sivertsen [2021b]. This section will only describe the highlights from that Notebook.

5.1.1 Basic data structure

As of *26-04-2022* the dataset consists of 41941504 entries. Most of the categories consist of roughly 1239 data points, depending on when the category was created. The oldest recorded data are from the beginning of 2019. It consists of 9 features. as shown in Table 5.1.

Table 5.1: Features of the Market insight dataset

Nr	Name	Type	Description
1	id	(int64)	Unique identifier
2	product id	(int64)	Associated Product id
3	manufacturer id	(int64)	Manufacturer id
4	cat id	(int64)	Associated category id
5	root cat id	(int64)	Associated root category id
6	date	(datetime64[ns])	The date the data was captured
7	hits	(int64)	How many times the product page was visited
8	clicks	(int64)	How many times users clicked to a retailer
9	last modified	(object)	When the row was last modified

It is worth noticing that hits and clicks are different features that measure the same thing: user interest. A hit is how many times the product page on Prisguiden.no is visited. A click is how many times a user follows a link from Prisguiden.no to an external retailer for a given product. A product can receive a hit and not a click if the user just visits a product detail page without clicking to a retailer. A product may receive a click without a hit if the user clicks on an AD or a campaign, which will lead the user directly to the retailer, skipping the product page on Prisguiden.no. The correlation between hits and clicks for all categories is roughly 0.578.

The dataset consists of 1325 unique categories and 310499 unique products. Each product is associated with a product category; for example, all CPUs are associated with the category *"Prosessor (CPU)"s*. Each product category is a leaf node of a category hierarchy. The category *"Prosessor (CPU)"* is a child of the parent category *"Datakomponenter"*, which itself is a child of *"Data"*.

Summing together all product clicks and hits to its closest parent category gives us a table on the format shown in Table 5.2.

5.1.2 Category plot analysis

Figure 5.2 show hits and clicks from a random sample of categories from 2019 to 2021. Most of the plots show a clear yearly periodic pattern, as shown in Figure 5.2a, Figure 5.2b, Figure 5.2c. The scale of values differs between categories. The category *"Mobiltelefon"* Figure 5.2b gets around 7500 hits per day. Meanwhile, the values within each category can differ vastly. *"Julekalender"* Figure 5.2c has a hit peak around 16000, while it usually gets around 0 hits per day.

Some categories experience extreme variations in traffic at different intervals. One example of this is the category *"Grafikkort GPU"*. The traffic tracked before

Table 5.2: Market Insights Overview dataset

cat_id	date	hits	clicks	product_id	cat_name
2	2018-12-02 00:00:00	2450	301	293349707	Bærbar PC
2	2018-12-03 00:00:00	2889	418	324468137	Bærbar PC
2	2018-12-04 00:00:00	3048	413	319458697	Bærbar PC
2	2018-12-05 00:00:00	2777	381	305158187	Bærbar PC
2	2018-12-06 00:00:00	2882	363	292147192	Bærbar PC
...		...	...	...	...
13771	2021-11-07 00:00:00	0	4	1061271	Fuglemat og meiseboller
13771	2021-11-08 00:00:00	36	6	3164575	Fuglemat og meiseboller
13771	2021-11-09 00:00:00	16	1	1572626	Fuglemat og meiseboller
13771	2021-11-10 00:00:00	19	10	2103281	Fuglemat og meiseboller
13771	2021-11-11 00:00:00	24	2	1572628	Fuglemat og meiseboller

and after Christmas 2020 differs vastly, and the data changes its behavior and distribution from then on.

Table 5.3 shows some basic statistics for *"Mobiltelefoner"*. This exact category was chosen at random just to get an idea of how one-time series might behave. The time series has a mean of 6453 hits and a standard deviation of 1881, which is around 30% of the mean. This is quite a big variance in the dataset.

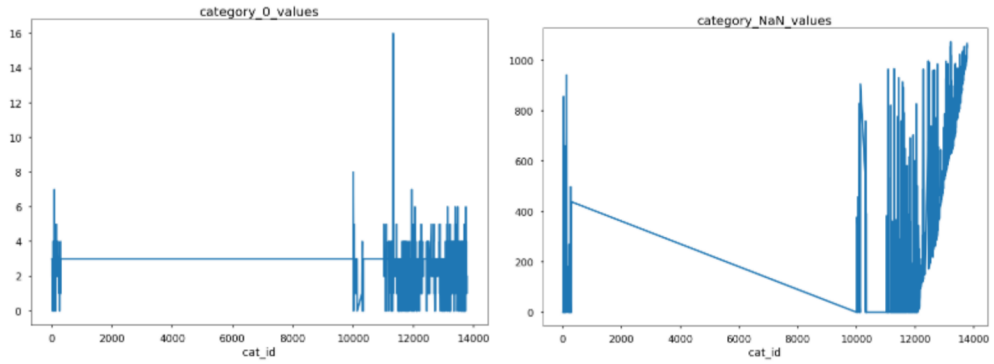
Table 5.3: Mobiltelefon statistics

	cat_id	hits	clicks
count	1074	1074	1074
mean	19	6453	1474
std	0	1881	649
min	19	0	0
25%	19	5672	1186
50%	19	6462	1388
75%	19	7339	1627
max	19	18699	10673

It is interesting to see how many categories receive zero values and undefined values each day. A zero value count in this context means how many days a given category has gotten 0 hits, but it has gotten some clicks. A NaN value in this context means that the given category has gotten 0 clicks and 0 hits on the given day. Figure 5.1 shows a lineplot of the different distributions. In general, rela-

tively few categories have 0-values. Some of the first categories created, the ones with id 0-500, there exists some days with 0 interest. However, most categories have at least one click and one hit each day. Categories with id above 10000 start to see a lot more zero values. The most likely explanation for this is that these categories are created at a later point in time. The steadily growing line at the end of Figure 5.1b near id 12000 supports this theory.

Figure 5.1: Counting how many categories have days with 0 hits or NaN values



(a) 0-values

(b) NaN-values

5.1.3 Correlation among categories

We did a correlation analysis to get an idea of how the categories correlate. We can create a correlation matrix by aggregating the dataset to a pivot table, using dates as the row index and the products id's as columns. To get reliable results, we limited the matrix to categories with more than 100 common data points. Because of the large number of categories, we created two matrixes. The small matrix in Figure 5.3, is made of a random subset of categories. This small one should be easy to read and understand. We also made a matrix of all the categories in Figure 5.4. This matrix is too complex to read details from but gives a full picture of the data. The lighter cells of the matrix indicate a correlation toward 1.0. It is worth noticing that the two matrixes do not use the exact same color spectrum. A completely white tile indicates not enough data to calculate a correlation.

Looking at the smaller correlation matrix in Figure 5.3 we can see that *"Hylle"* correlates strongly with *"Bord"* with a corraltion of 0.7. *"Ryddesag"* and *"Kontrollenhet og gateway"* does not correlate at all, with a correlation of 0.04. *"Singlet*

til barn" does not have enough data yet, so the whole row is white, indicating missing values.

Based on the matrix's color spectrum of the full correlation matrix in Figure 5.4, it is clear that categories cover almost the whole spectrum of correlation relationships. The overall bright colors indicate a bias towards positive correlation.

Also worth noticing from the big correlation matrix is an indication of more purple lines and areas to the lower right corner of the matrix. Tech-related categories dominated Prisguiden's product category in its early years. In recent years they have expanded their reach to include more common household products. It makes sense that these new products do not correlate much with technology products.

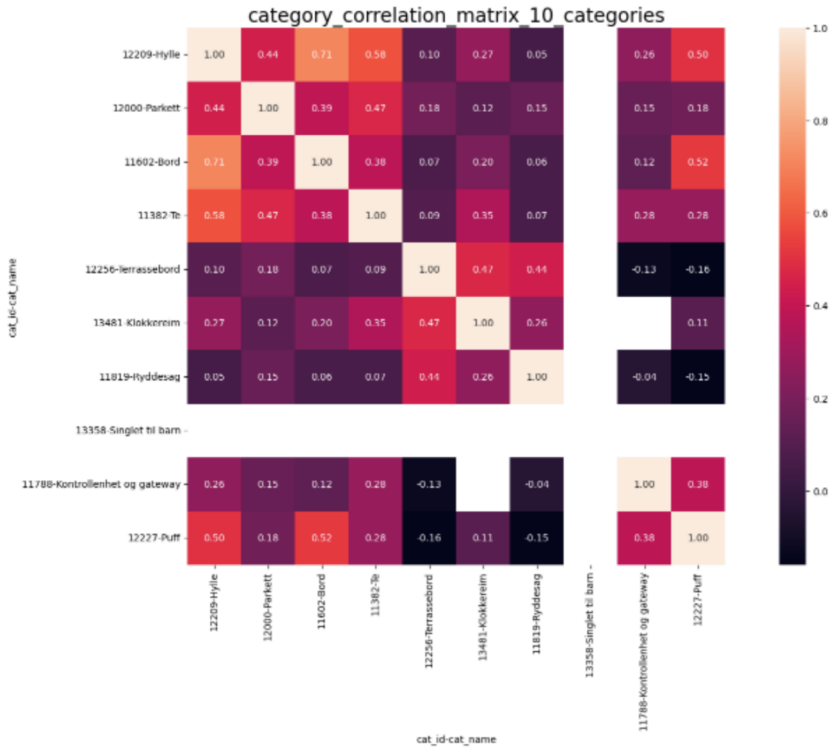


Figure 5.3: Correlation matrix of 10 random categories

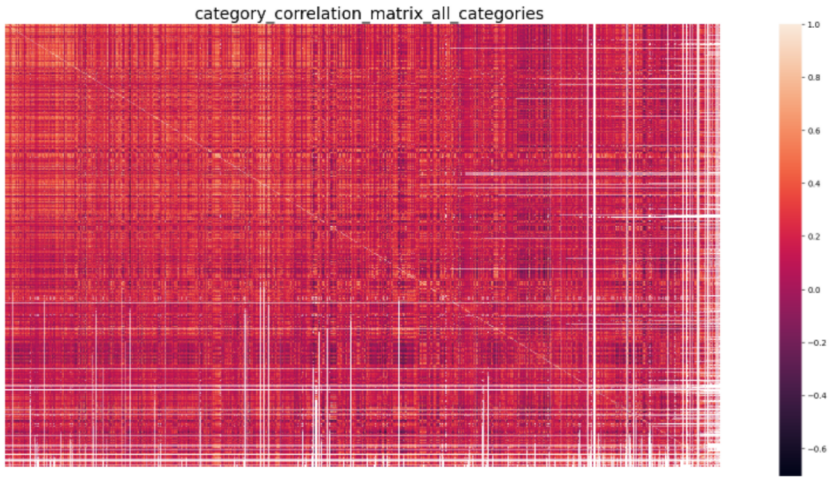


Figure 5.4: Full category correlation matrix

5.2 Datasets

The complete dataset supplied by "Prisguiden.no" contains over 1300 unique product categories. To achieve product trend predictions of categories the dataset is split into smaller subsets of data to be analyzed. These datasets create the basis for the experiments to be conducted. Initially, 3 datasets were selected, each with its properties and connectivity. The first two datasets are created based on correlation. The first dataset is defined with 20 highly correlating categories. Dataset 2 is comprised of 20 categories with very little correlation among them. This is done to evaluate if a global model will improve predictions when trained across related time series.

Dataset 3 contains 8 categories picked by domain experts at Priguiden.no, and it contains highly seasonal data with high extreme values.

5.2.1 Dataset 1 - Correlation

A dataset with a high correlation between the categories is selected to use the correlation to improve the model predictions. The hope is that the correlating data gives global models more relevant data to work with. This can then be compared to a non-correlating dataset (dataset 2) to evaluate the importance of the data correlation in this case.

To select correlating time series as part of the correlating dataset, a combination of manual data selection through domain knowledge and auto-correlation was used. A method for calculating auto-correlation was used to create a heat

map of correlation between all the categories. Category autocorrelation was calculated using Using this heat map [Figure 5.5], we were able to evaluate what clusters of categories had the highest concentration of autocorrelation. In order to make correlating datasets stand out more clearly, we remove values from category pairs with an auto-correlation lower than 0.5 across the categories and a constraint of minimum 100 observations needed.

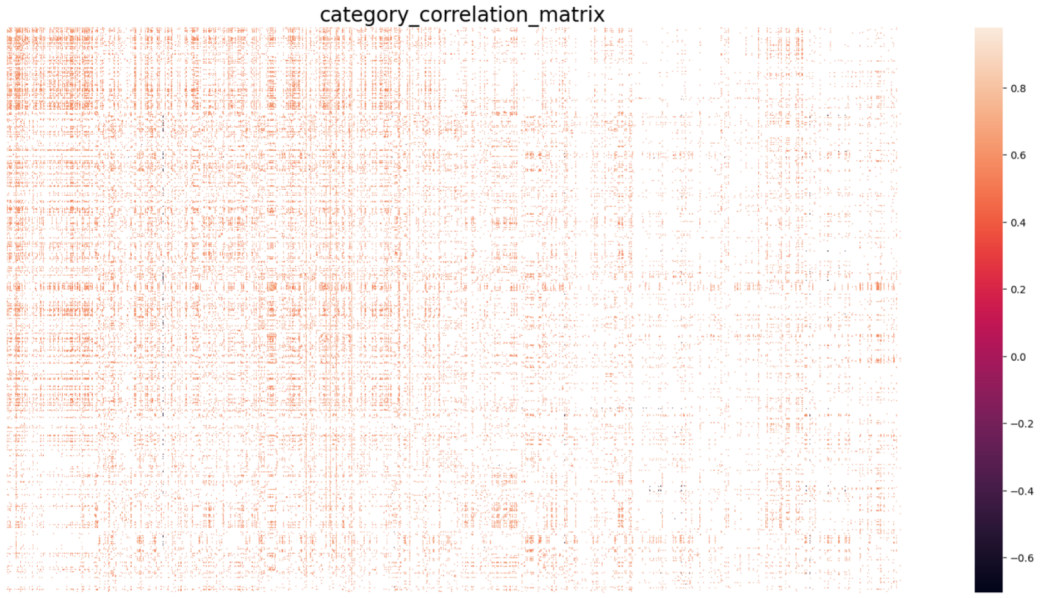


Figure 5.5: A correlation matrix showing the autocorrelation of all categories, filtered out values of autocorrelation below 0.5 in order to increase visibility of high correlation values and groupings.

Using the heatmap as a reference, it was clear that the interval of the first 30 or so categories has a high autocorrelation. Additionally, the categories with the highest correlation are all categories contained within the domain of electronics. Using domain knowledge we are able to assert that these categories should have a higher correlation between them than other categories probably have. With these two assumptions comprising the basis for the selection, we were able to filter out categories with less correlation, resulting in a set of 20 categories comprising the correlating dataset.

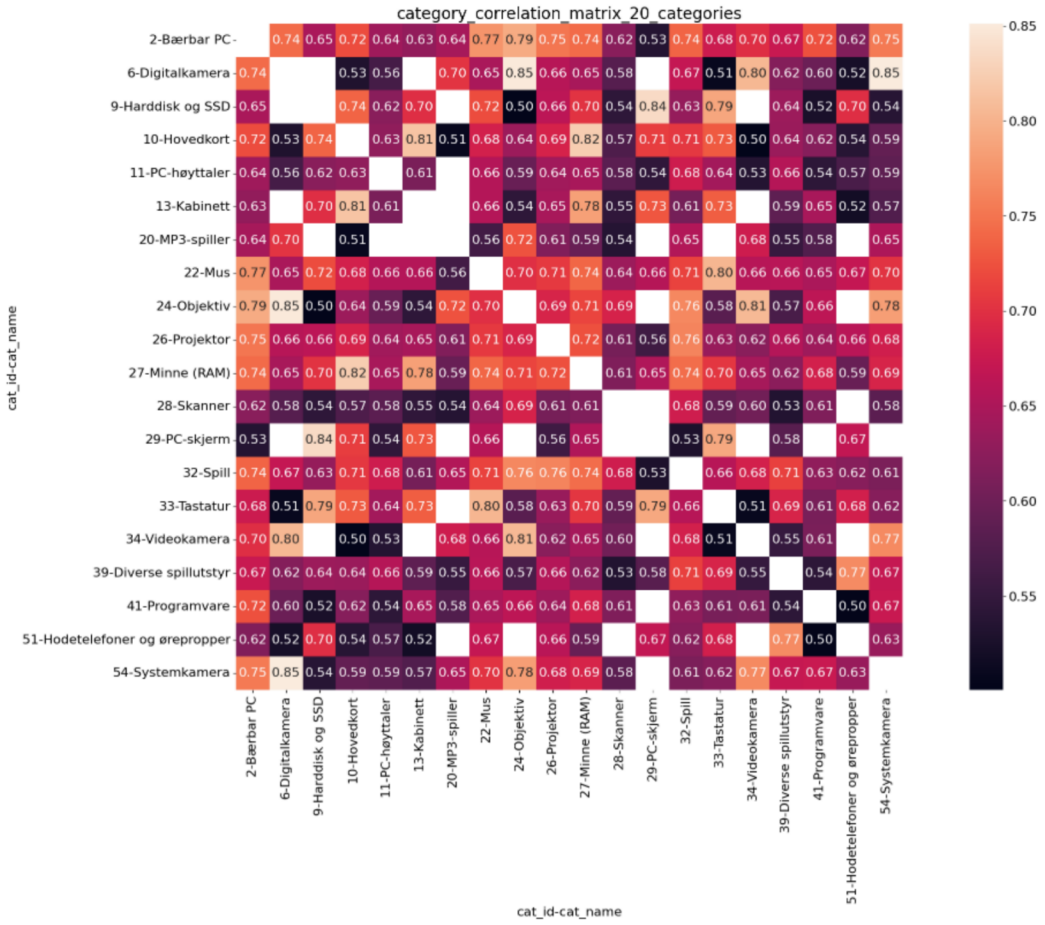


Figure 5.6: A correlation matrix showing the autocorrelation between the selected 20 categories of the highly correlating category dataset - Dataset 1.

The filtered dataset contains 20 categories from the electronics domain, some of the most highly correlating datasets in our dataset. The selected categories for dataset 1 are shown with a heatmap in Figure 5.6, and listed in Table 5.4.

Table 5.4: Selected categories comprising dataset 1 - Correlating categories

Category ID	Name(Norwegian)
2	Bærbar PC
6	Digitalkamera
9	Harddisk og SSD
10	Hovedkort
11	PC-høytaler
13	Kabinett
20	MP3-Spiller
22	Mus
24	Objektir
26	Prosjektor
27	Minne (RAM)
28	Skanner
29	PC-skjerm
32	Spill
33	Tastatur
34	Videokamera
39	Diverse spillutstyr
41	Programvare
51	Hodetelefoner
54	Systemkamera

5.2.2 Dataset 2 - No Correlation

As the second dataset, a collection of 20 categories with the lowest possible correlation is defined. A dataset with close to no natural correlation either through autocorrelation or through the use of domain knowledge, the second dataset is defined in order to serve as an opposite to the first dataset. This dataset is meant to serve as an opposite to the first dataset, making it easier to evaluate if the correlation between categories has any significant effect on forecasting.

To find a set of 20 categories with the lowest possible correlation, we conducted a search on the values of calculated auto-correlation. Using auto-correlation analysis on the dataset, the categories were sorted after the amount of correlation summed across all categories. Out of the sorted list, the top 50 categories with the least correlation were selected. Of these 50, seasonal categories such as product categories related to Christmas were filtered out. After this, a manual selection was conducted on the remaining categories in order to remove those with a high correlation between the remaining categories. This was repeated until only 20 categories remain.

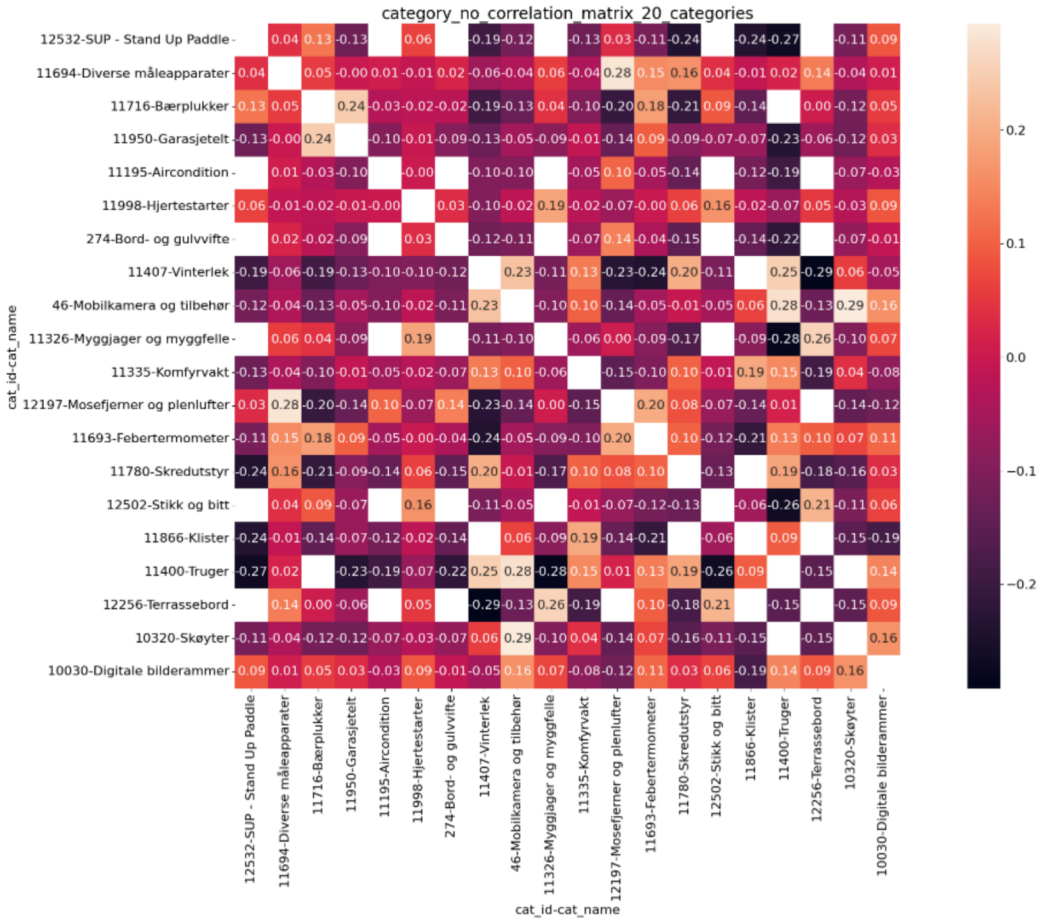


Figure 5.7: A correlation matrix showing the autocorrelation between the selected 20 categories with the lowest autocorrelation - Dataset 2.

The filtered dataset contains 20 categories with the least inward correlation we could find. The selected categories for dataset 2 are shown with a heatmap in Figure 5.7, and listed in Table 5.5.

Table 5.5: Selected categories comprising dataset 2 - Non- correlating categories

Category ID	Name(Norwegian)
12532	Hodetelefoner
11694	Diverse måleapparater
11716	Bærplukker
11950	Garasjetelt
11988	Aircondition
11998	Hjertestarter
274	Bord- og gulvvifte
11407	Vinterlek
46	Mobilkamera og tilbehør
11326	Myggjager og myggfelle
11335	Komfyrvakt
12197	Mosefjerner og plenlufter
11693	Febertermometer
11780	Skredutstyr
12502	Stikk og bitt
11866	Klister
11400	Truger
12256	Terrassebord
10320	Skøyter
10030	Digitale bilderammer

5.2.3 Dataset 3 - Seasonality

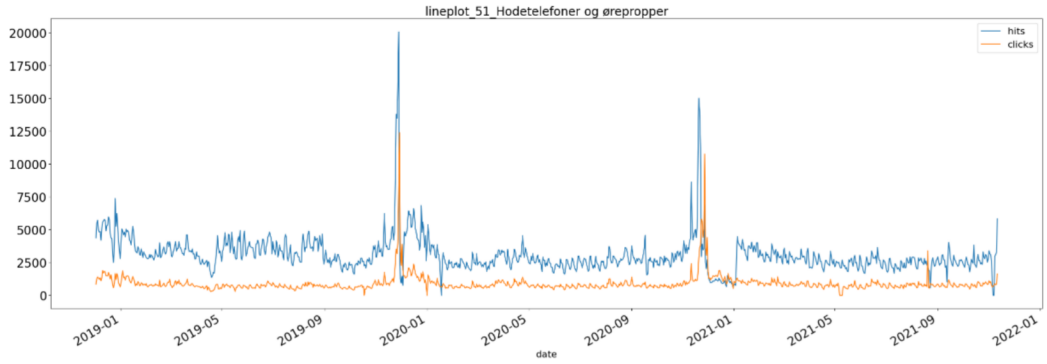
Dataset 3 is selected due to the high seasonality contained within the time series. The dataset consists of 7 time series with high seasonality. While datasets 1 and 2 were selected based on correlation analysis between the time series, dataset 3 is found through the application of domain knowledge.

After talks with contacts at "Prisguiden.no" a list of 8 highly seasonally dependent time series were found. This list of products contains categories heavily dependent on seasons, such as Christmas and summer. A list of the time series contained in dataset 3 can be found in Table 5.6.

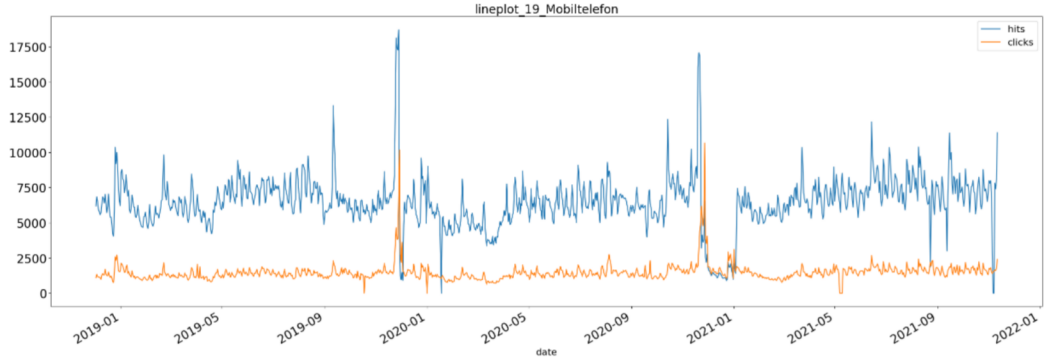
Table 5.6: Selected categories comprising dataset 3

Category ID	Name(Norwegian)
12322	Vinterjakke
11428	Vintersko
11850	Langrennski
11852	Skisko
273	Varmeovn
11036	Snøfreser
11213	Snøskuffe
12532	Varmepumpe

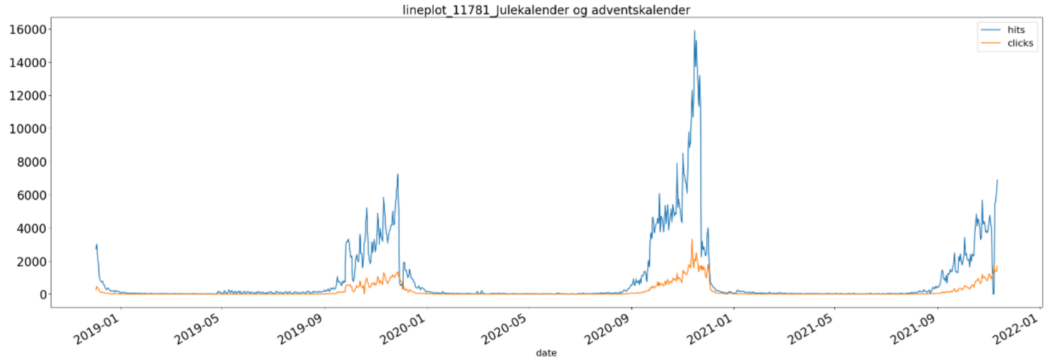
Figure 5.2: Category plots of hits and click rate from 2019-2021



(a) Hits and clicks rate for *"Hodetelefoner og øreppropper"*



(b) Hits and clicks rate for *"Mobiltelefon"*



(c) Hits and clicks rate for *"Julekalender"*

Chapter 6

Method

The applied methodology used in this thesis is explored in this chapter. Section 6.1 start with presenting the hardware used for running the experiments. Section 6.2 then presents the error metrics used to measure the accuracy of the predictions across models. Running experiments are done through the use of the framework developed. This framework is explored in more detail in Section 6.3.

Data processing steps used to manipulate data before the use with models are presented in Section 6.5.

Model methods are then defined and explored. The SARIMA model is described in Section 6.6, the LSTM model in Section 6.7, and the hybrid model CNN-AE and LSTM in Section 6.8. Lastly, Section 6.9 discusses the use of the t-test for testing the statistical significance of results.

6.1 Software and Hardware

The framework and experiments are implemented using python 3.8 [https://www.python.org/]. The SARIMA method is created using the pmdarima [https://pypi.org/project/pmdarima/] library and statsmodels library [https://www.statsmodels.org/stable/index.html], supporting both running the experiments and tuning with auto-arima from the pmdarima package. On the other hand, the machine learning library Keras is used to implement the deep learning methods used such as the LSTM method [https://keras.io/].

When loading data into the experiments, data pipelines were created using the genpipes library [https://pypi.org/project/genpipes/]. After this, data manipulation was conducted using the Pandas library, as well as the numpy library [https://numpy.org/].

The hardware available for running experiments consists of two work-stations.

The first work-station consists of a AMD Ryzen 5 5600X processor and 32 GB 32000 MHz memory with Windows 11 and Linux Sub-system for Linux, while the second work-station consists of an Intel i9-9900K processor and 16 GB 2666 MHz memory, Windows 10 and Linux Sub-system for Linux. Both of these work-stations were used for tuning and execution of experiments in this project.

All of the code are open and available at [Github Sindre Sivertsen [2022]].

6.2 Loss function and Metrics

Here the used loss function and metrics are described.

Loss function

Since our datasets often contain a lot of outliers we decided not to use the well known Mean Squared Error (MSE) [Section 2.1.7]. Instead, we used Mean Absolute Error (MAE) while training the neural networks. We can use the MAE during training because of the data-preprocessing, which normalizes the time series values. Some experiments were done using both MASE and sMASE, but MAE did perform better.

Metrics

When choosing an error metric, we have to accommodate a number of factors. Since our time series are of different scales, we cannot use a scale-dependent metric like MSE or MAE on our test data, as the test data is never touched, each time series might be on different scales.

We use MASE with 1-day naive forecast, as well as MASE with a 7-day naive forecast. This is because the 1-day naive forecast is a regular metric used and can give a good impression of how the model performs. However, a 7-day naive forecast will more closely represent a real-world application measure, because if a model gets a score higher than 1, it means that it has no real world application, and it's better to use the previous week as a forecast for the next week.

A problem with MASE is that if a time series follows a random walk then the best forecast will always be the naive forecast. MASE is therefore dependent on how good the naive prediction is. Therefore we also include sMAPE as a metric. sMAPE will give a better impression of how well our predicted forecasts fit the target values, independently of the naive forecast. In the chosen datasets zero values rarely occur. Therefore, the sMAPE metric can be used without worrying about zero divisions [Section 2.1.7].

6.3 Experiment Framework

In order to make it easier to execute multiple experiments, which all saved enough metadata for the experiment to be recreatable and repeatable we made a framework. It can be deconstructed into four main modules. The configuration module, a data processing module, an experiment module, and the save experiment module.

6.3.1 Pipeline module

The data processing module has two functions. The first one is to streamline all data processing steps and structure them in one shared place. Secondly, to have a self-documenting data module that can easily save every processing step applied to the data before the experiments are executed. Examples of a pipeline steps output are shown in Table 6.1 and in Table 6.2.

Table 6.1: Base data processing steps

Step	Description
1	load market insight data and categories and merge them
2	convert date columns to date_time format
3	sum up clicks to category level [groupBy(date, cat_id)]
4	rename column 'title' to 'cat_name'
5	combine feature 'hits' and 'clicks' to new feature 'interest'
6	drop columns 'hits' and 'clicks'
7	filter out data from early 2018-12-01
8	drop uninteresting colums

Table 6.2: LSTM data processing steps

Step	Description
1	Convert input dataset to generator object
2	filter out category
3	choose columns 'interest' and 'date'
4	fill in dates with zero values
5	convert to np.array
6	scale data using standardization
7	generate x y pairs with sliding window with input size 10, and output size 7
8	generate training and validation data with training size 7

Multiple pipelines were created in order to support all the different experiments. While the deep learning methods such as the LSTM and the hybrid model only need two shared pipelines, one for univariate and one for multivariate, other experiments such as the SARIMA is in need of other pipelines. These pipelines are designed with the aim of modularity, making the expansion of multiple pipelines with multiple shared steps easy.

6.3.2 Config module

The configuration module is created with the aim of making model selection and configuration more easily accessible. Using separate configuration files for model configurations, the amount of code that needs to be changed for the different experiments is reduced. Additionally, by logging the configuration of models for each experiment, backtracking the results of different configurations are more accessible.

The configuration is done through the use of two separate ".yaml" files. The first one is the "config.yaml" file. This contains information that is shared among all of the different experiments, such as the selected data files, random seeds, logged error metrics etc. Secondly, a separate configuration file contains the information more relevant to the specific experiment. These configuration files contain information such as the selected predictive model for the experiment, as well as the model hyperparameters and tuning parameters.

During runtime, the two configuration files are merged, adding the information from the model configuration to the more general experiment configuration. The config is parsed through at runtime, accessing the needed information when it becomes relevant. A config example is available in the appendix at Section 8.6. Configs are also available in the source code.

6.3.3 Save Experiment module

The save source module handled everything related to logging and saving an experiment. Each experiment is associated with a unique descriptive ID and a description text to easily differentiate between experiments. Each experiment saves trained models, configs, the dataset used, stdout logs, training metrics, validation metrics, testing metrics and figures.

Everything is save to the local "./models/" folder, and to an exernal ML experiment tracking tool named *neptune.ai*. The Neptune experiments are public and can be seen at *https://app.neptune.ai/sjsivertandsanderkk/Masteroppgave/*. The local folder structure of a saved experiment is shown below.

```
arima-predict-cat-11037-7-days
├── Arima-11037.pkl
└── data-processing-steps.txt
```

```
├── datasets.json
├── figures
│   ├── 11037-Data-Prediction.png
│   ├── 11037-Predictions.png
│   ├── 11037-Training-data-approximation.png
│   └── 11037-Training-data.png
├── logging
├── training-errors.csv
├── metrics.txt
├── options.yaml
├── predictions.csv
├── tags.txt
└── title-description.txt
2 directories, 13 files
```

6.3.4 Packages and verions

Table 6.3: Experiment Python packages and versions

Package	Version
matplotlib	3.5.1
matplotlib-inline	0.1.3
numpy	1.22.2
pandas	1.4.0
pandocfilters	1.5.0
sklearn	0.0
statsmodels	0.13.1
torch	1.10.2
optuna	2.10.0
plotly	5.6.0
pytorch-lightning	1.5.10
keras	2.8.0
tensorflow	2.8.0
tensorflow-io-gcs-filesystem	0.24.0
optkeras	0.0.7
pmdarima	1.8.5

Table 6.3 show the most important python packages used for the experimental setup and their respective versions. The full list of packages and versions are supplied in the *requirements.txt* file at the github code repository Sindre Sivertsen [2022]

6.4 Locking Random Seed for reproducability

In order to make our experiments reproducible, the first code that runs at the beginning of each experiment is the code that locks a pre-defined random seed number on all our packages that have stochastic functions. The random seed is defined in our config [Section 6.3.2], and was always set to 42.

The list of packages we lock the random seed on are as follows:

- *random.seed*
- *os.environment*
- *np.random.seed*
- *torch.manual_seed*
- *torch.cuda.manual_seed*
- *torch.backends.cuddn.deteriministic*
- *tf.random.set_seed*

6.5 Data preprocessing

This section briefly describes all of the data processing done on the datasets defined in Section 5.2.

6.5.1 Train, validation, test splitting

The inspiration for the training, validation and test splitting is taken from Bandara et al. [2019], Hewamalage et al. [2021]. Both reserve the last m sized part of the training set as a validation set, where m is the forecasting window size. However, Hewamalage et al. [2021] note that this is problematic. This is because the last part of the training set is removed and therefore not considered when training the model, as it is used only for validation during model selection and tuning.

The further away the test predictions are from the training set the worse, because the underlying patterns may change during this last part of the sequence. Therefore Hewamalage et al. [2021] suggests an alteration to the use of the validation split. The validation set is split from the training set only when tuning the model and is thereafter added to the training data when the model is used and tested. Thus, the models will be trained on all available data, excluding only the selected test set.

We did however make some modifications to the approach presented by Hewamalage et al. [2021]. Initially, the complete dataset is available. A test set, consisting of the last m points of the sequence, is then manually split from the complete dataset. After this, the training set is split again to define both the used training set and the validation set. For the SARIMA method, a validation split is not needed. However, the use of a LSTM, specifically a stateful LSTM, causes restrictions to the creation of the validation set. Due to the use of stateful LSTM modules, the validation set has to be the same batch size as the training set. This, in turn, means that using a validation set equal to the forecasting window m would limit the available batch size for the model used on the training set.

Using a validation set equal to one forecasting window, also means that if the model is only trained on one-time series, the validation set would have exactly one sample in it. This might not be problematic if the time series is homogeneous, but if the pattern in a time s series varies it can, because the one validation sample might not be representable for the whole series.

Bandara et al. [2019] would always train their models on multiple time series, which meant that their validation set would be $validation\_size = m * number\_of\_series$. During development and testing, it was found that a validation set size of only one forecasting window is exceedingly small, in our case. Tuning on validation error would be severely limited and would produce models with less optimal hyperparameters.

As a result of these findings, we selected a batch size for the training set beforehand, then used the chosen batch size as a validation set size.

This, however, means that we are not able to conduct a hyperparameter search for an optimal batch size because that would change the validation set as well. When we tried to tune the batch size with a variable validation set, the model would almost always prefer the smallest possible batch size. This is because it would result in a small validation set, thus reducing the number of possible errors.

When tuning the models, the validation set is actively used as a measurement of the predictive ability of the model. However, when conducting final experiments with well-defined models and parameters, the validation set is once again added to the training set so that the model has more data to train on. This is the same approach as used in Hewamalage et al. [2021].

Figure 6.1 shows an illustration of how one dataset is split up during hyperparameter tuning. The whole time series has a size of length L. The test set T is taken from the end of the series and has the size of the forecast window $L = m$. The validation set V has the size of one batch size $V = batch\_size$. So during tuning, the size of the training data is $T = L - (V + T)$. During testing the validation set V is re-added to the training set, making the training set $T = L - T$.

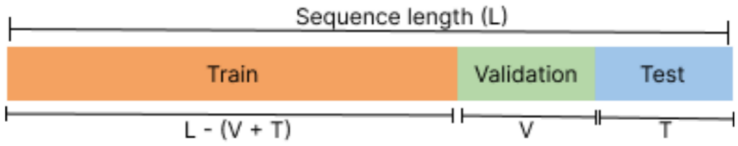


Figure 6.1: Illustration of trainig, validation, and test split. V = batch size, T = m = forecast window

The exact time procedure is done when training global models across multiple time series. The train-val-test-split process is repeated for each time series in the set, and then the training data for the different time series are concatenated into one long training sequence. The same is done to the validation data and test data respectively. Figure 6.2 show how time series are concatenated.

Figure 6.2: Illustration of how multiple time series are concatonated for the global method

6.5.2 Feature Engeneering

Both the feature *"hits"* and *"clicks"* can be argued to measure user interest. A product's hits score measures how many times someone has clicked on that product information page at prisuigen.no. A product's click score to measure how many times prisguiden.no has redirected a user to an external retailer for the given product. We ended up combining these two variables into one feature which we called **"interest"** with the equation defined in Equation (6.1).

$$interest = hits + clicks \tag{6.1}$$

6.5.3 Value Scaling

Deep learning models require appropriate scaling to converge properly. For the neural networks, we scale the data using standardization as shown in Equation (6.2).

$$z = \frac{x - \mu}{\sigma} \tag{6.2}$$

Where μ is the mean and σ is the standard deviation.

We also tried to normalize the data by scaling the values down to a predefined fixed range [0.1, 1] using Equation (6.3). However, during experimentation and testing, it was found that scaling with standardization outperformed the predictions made with a normalization scaling. Both methods were tested, but at last, standardization was selected as the preferred scaling method.

$$x\_std = \frac{x - x.min()}{x.max() - x.min()} \tag{6.3}$$

$$x\_scaled = x\_std * (h - l) + l \tag{6.4}$$

where h is the max feature range and l is the min.

6.5.4 Univariate to Multivariate feature engeneering

Since the E-commerce domain is heavily influenced by external factors, such as holidays, Christmas, and yearly seasons we thought it would be beneficial for the neural net models to be aware of which day of the week it is, which month it is, and if it is closer to summer than winter.

We add the *month* feature, a number between [0, 11] depending on which month.

We add the *day* feature, a number between [0, 6] where 6 is saturday, 0 is sunday etc. We can calculate the day of the week using the formula Equation (2.15) from Section 2.1.15.

We add the feature *season* which is a number between [-1, 1] using Equation (6.5).

$$season = \cos(\pi * (month \mod 12)/6) \tag{6.5}$$

Figure 6.3 shows the output of the season feature. If the month is close to Christmas (December, January) this feature will be close to 1. If the month is closer to summer (May, June) this feature will be close to -1.

In the and all added features are scaled between [0.1, 1] as described above.

6.5.5 Moving Window Approach

When creating the datasets, we apply the same moving window approach as applied by Bandara et al. [2019] and Hewamalage et al. [2021]. This strategy transforms a time series X_i into pairs of $< input, output >$ patches. In other words, we transform the problem into a supervised learning problem. [Figure 6.4] show an illustration of the moving window approach.

Given a time series $X_i = \{x_1, ..., x_K\} \in \mathbb{R}^K$ of length K a mowing window algorithm will convert the time series X_i into $K - n - m$ number of patches,

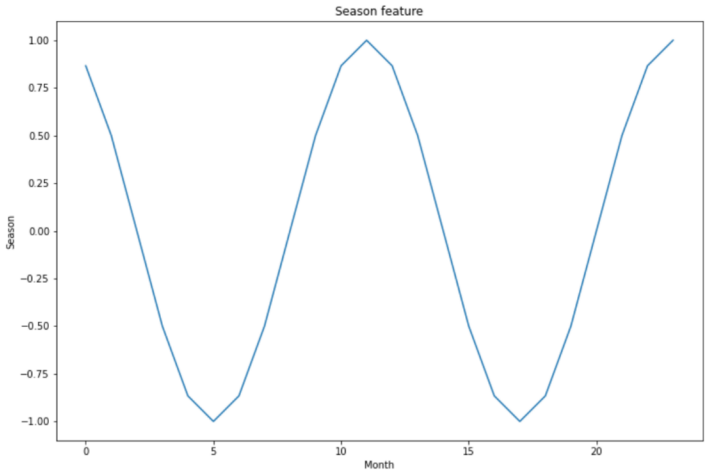


Figure 6.3: Season feature

where each patch has the size of $m + n$, and n is the size of the input window, and m is the size of the output window.

This thesis will primarily work with output window m for size 7. This is because it is safe to assume that the E-commerce domain consists of weekly patterns. 7 days is short enough that we should be able to forecast something meaningful accurately but long enough to have some commercial value.

Hewamalage et al. [2021] suggests setting the input window size n to be slightly bigger than the input window with $m * 1.25$. Or putting m slightly bigger than the seasonality period $m = 1.25 * seasonality\_period$. We selected to make the input window a bit bigger than the output window and the weekly period, by setting $m = 10$.

6.5.6 Modeling Trend and Seasonality

The literature is conflicting when it comes to NN's capability to model trends and seasonality. Sharda and Patil [1992] work inferred that NNs are capable of modeling seasonality accurately. However, more recent work suggests that deseasonalisation and detrending can boost the performance of neural networks [Zhang and Qi [2005] Smyl [2020]]. Ouyang et al. [2021] show that detrending and deseasonalisation using STL decomposition will harm machine learning models, but improve statistical models.

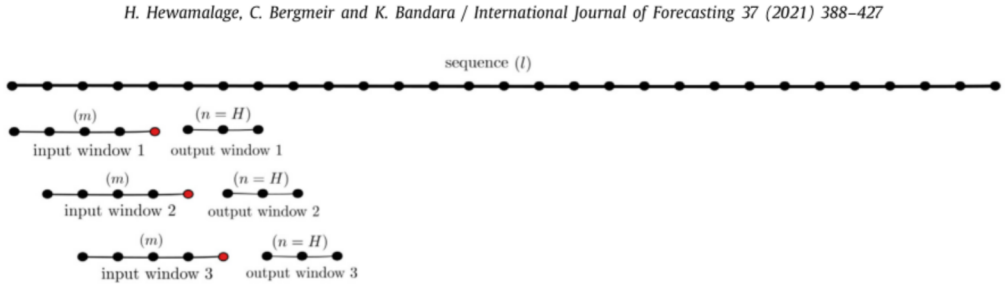


Figure 6.4: Moving window scheme [Hewamalage et al., 2021]

Hewamalage et al. [2021] conclude that when all the series in a dataset follow homogeneous seasonal patterns, with all of them covering the same duration in time with sufficient lengths, RNNs are capable of capturing seasonality without prior deseasonalisation. If this is not the case then RNNs have a hard time modeling seasonality on their own, and deseasonalisation step should be employed.

Montero-Manso and Hyndman [2021] advocates for not removing seasonality on global models based on their empirical results. They conclude that global NNs do not suffer from an inability to pick seasonal patterns, as long as the models are sufficiently complex and have enough memory. They argue that removing seasonal patterns from the data can erase some relevant information that might be exploited by global models.

These conflicting results in the literature, combined with our varied set of time series we conclude that we will avoid applying STL decomposition in our main experiments, but might do some experiments to see how it can improve our metrics.

6.5.7 Scaling down outliers

Some of our time series consists of a few extreme outliers. An example of how one of these outliers might occur is if a supplier makes an error and sends a wrong price to Prisguiden. If this price is much lower than the original price then it might spike a huge user interest for this item, until Prisguiden finds and corrects the error. These outliers do not give any meaningful information, so steps were taken to reduce these extreme values.

We use the standard deviation as a base to scale down the outliers with the formula below.

$$y(t_n) = \begin{cases} x * 0.3 & \text{if } x \geq standard\_devation * 5 \\ x, & \text{otherwise} \end{cases}$$

An example of how this affects a time series is shown in Figure 6.5.

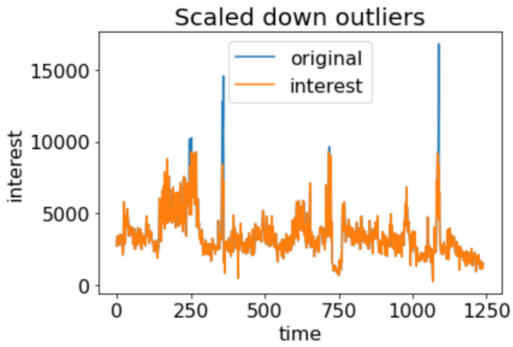


Figure 6.5: Illustration of how the scaling effects a time seires. The blue is the original series. The orange is the processed series.

6.5.8 Data Processing Summary

So far, we have described each processing step and how it is conducted, but not the overall order.

The main preprocessing steps was carried out as follows:

1. Combine *hits* and *clicks* to feature *interest*.
2. Split up into test and training data.
3. If a multivariate model $->$ Generate date features.
4. Scale down outliers
5. Normalize the data
6. Generate sliding window patches
7. Split the training set into training and validation set.

If the model is a global model. Then the same steps described above are executed once for each time series, and then the training sets, validation sets, and test sets are concatenated.

The post-processing of the predicted forecasts is just a matter of reversing the local normalization to the forecast.

6.6 SARIMA baseline

The SARIMA model is the first model to create baseline metrics for comparisons with other models. The SARIMA model is limited to being a local and univariate method, thus limiting the number of experiments that can be run.

Due to the inherent seasonal component present in the E-commerce data available in this thesis, the seasonal SARIMA model is selected in favor of the non-seasonal ARIMA model.

The SARIMA parameter values are shown in the appendix in table Table 8.2.

SARIMA Train, Test splitting

Using the auto-arima method supplied by the pmdarima python library, there was no need for splitting training data into a training set and validation set. The auto-arima method requires only the input of one training set.

However, the creation of a test set is required. After the models are created and tuned, they are used to make predictions, which is then compared to the test set. The test set is then used to measure the predictive error and accuracy of the models.

6.6.1 SARIMA Tuning

Model tuning of the SARIMA model is done with the auto-arima framework. Auto-arima utilizes Bayesian optimization to find a well-suited set of hyperparameters to use, within a defined search space for parameters. Using the MAE metric as a measure of performance, auto-arima search for a set of parameters for each individual time series in the datasets defined in Section 5.2. The parameter search space used during the tuning is defined in Table 6.4 with the range of parameter values.

Table 6.4: Parameter search space auto-arima, SARIMA tuning

Parameter	Min	Max
p	0	7
q	0	7
d	0	7
P	0	5
Q	0	5
D	0	5
S	12	12

Through testing of the SARIMA model, the parameter range of 0 to 7 for

values p,q, and d was found. The same is the case for the P, D, and Q values with the range from 0 to 5. In order to counteract the likelihood of creating a highly overfitted model, the values were limited to a range of 7 and 5 due to results from testing the models.

The seasonal component S is selected to be 12, assuming a monthly seasonality.

6.7 LSTM

The LSTM consists of stacked layers of LSTM cells followed by a dense layer with the same output size as our forecast horizon. The parameters used for the LSTM cells are listed in Table 8.3 and the parameters used in the dense layer are listed in Table 8.4, both found in the appendix Section 8.7.

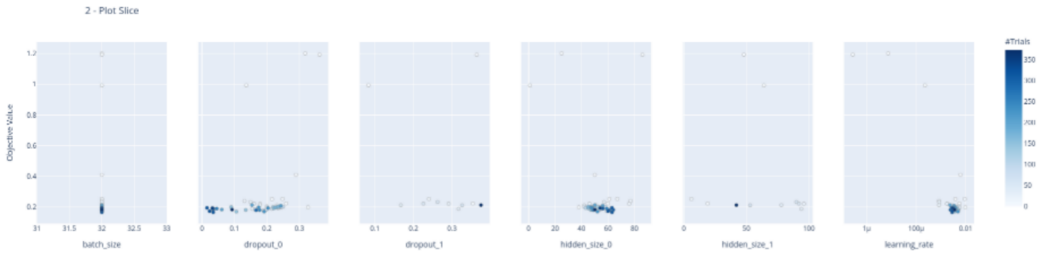
LSTM Tuning

The hyperparameters of the LSTM are tuned with the Optuna Python package with the Bayesian optimal algorithm [Section 2.1.8]. The hyperparameter minimum and maximum values are shown in Table 6.5. These values were carefully chosen after many tuning experiments. Figure 6.6 show a report from an optuna tuning, and how different parameter combinations improved the model.

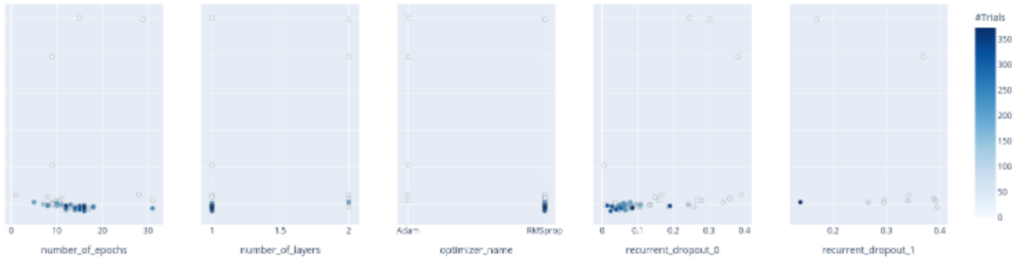
Table 6.5: LSTM Hyperparameter tuning range

hyperparameter	Tuning range
hidden_size	[1, 100]
number_of_layers	[1, 2]
dropout	[0.0, 0.4]
recurrent_dropout	[0.0, 0.4]
optimizer_name	['RMSprop', 'Adam']
learning_rate	[1e-7, 1e-2]
number_of_epochs	[1, 40]
batch_size	[32, 32]
input_window_size	10
output_window_size	7
number_of_features	1
number_of_trials	200
stateful_lstm	true

Figure 6.6: A Optuna generated SLICE plot showing model perfomance with different hyperparameter combinations.



(a) First 6 hyperparameters



(b) Last 5 hyperparameters

Stateful LSTM and hidden states

Intuitively the problem at hand requires a stateful LSTM [Section 2.1.13]. One problem with stateful LSTMs is that they require all the input to have the same batch size. This is because the two-state vectors h_t and c_t have the shape $(batch\_size, input\_length, features)$. In a stateless LSTM, these vectors are initialized with zero values at the beginning of each forward pass, so the *batch_size* can be inferred from the input. In a stateful LSTM, these vectors are initialized once together with the rest of the network, and so the *batch_size* needs to be specified with the rest of the hyperparameters.

As a consequence the training data, validation data, and test data all need to use the same *batch_size*, which is problematic. The easiest solution is to use online learning, by setting the batch size to 1. However, this means calculating gradients and doing a backpropagation for each timestep in our dataset, which is highly inefficient.

A batch size bigger than one means that the number of samples in the training data and validation data and test data has to all be dividable by the batch size. If not the last batch will have too few samples in it, and the network will reject it. The most common solution is to find the highest common factor between the training data and the test data, and use that as batch size. This is not a solution that works on this problem because in our global model we train our model on many different time series of different lengths, and also, our test data consists of exactly 1 sample with a size equal the forecasting horizon, per time series. We managed to solve the test data problem by creating two models with different batch sizes. We trained the first model on the training data, then copied the internal weights over to the second model, which had a batch size of one, for testing and predictions.

The solution of copying weights was not available for the validation data because the validation step in Keras is incorporated into the training function. Due to time constraints, we did not want to dive deep into the Keras internals to see if it was possible to solve this problem by hand.

Our solution ended up being to remove a number of samples at the beginning of the dataset in order to make the total number of samples dividable by the batch size, and then make the validation set equal to one batch. Removing data from the beginning, seemed obvious as newer data will have more relevance than older data. This meant that we also had to remove batch size as a tunable hyperparameter, because the model would favor lower batch sizes because that meant fewer validation data batches, which equals potentially fewer examples to get wrong.

Instead of automatically tuning batch sizes, we did some manual testing, and ended up setting batch size 32 for all experiments.

Reset states

Since the hidden states are never reset automatically in Keras stateful LSTM models. Therefore, we had to do it ourselves.

Local models

For the local models, we are resetting the internal state at the beginning of each epoch as well as before testing. We do not reset states before validation, because the validation step always comes after an epoch run, and the validation data are taken from the end of the training data for each time series. This means the model will in fact have the correct hidden state when starting the validation step.

During testing, we have to be a bit more clever. We ran the whole training set plus the validation set through the networks prediction loop before predicting the test set and calculate test metrics. This is for the model to "see" the whole picture and have the correct internal state before predicting the test set.

Since we track all experiments we ran, as described in Section 6.3.3, we can reference some of our testing results to back up our choices. Neptune experiment *MAS-396* and *MAS-397* confirms that this tactic pay off. *MAS-397* achieved a MASE score of 2.00 on the test set without resetting states and passing through training and validation data before the test set. In *MAS-396* we used the exact same setup, but with resetting states before the test set. MAS-396 achieved a MASE score of 1.67.

Global models

For global models, we followed the concept presented by Smyl [2020], with global weights, but local states.

As described in Section 6.3 we concatenate multiple time series after one another during the training of a global model. This means we have to reset the hidden state during the training of a global model or else we will give our model a false sense of dependencies between the independent time series.

As described in Section 6.7 all batches had to be of equal size. Our time series were concatenated one after another into one large time series. Our solution was to pass in a lambda callback to Keras which ran after each batch. The number of batched needed to fit each time series was counted. We then counted how many batches it had executed. When the number of batches executed equals the number of batches in a time series we reset the hidden state. However, this created a flaw in experiments with more than one time series. Hidden states might not reset at the exact right time, but in the worst-case scenario, a $batch\_size - 1$ number of samples too late. Still, this technique did prove to be successful. Neptun experiment *MAS-491* achieved a MASE metric of **19.61** without resetting states during training. Neptune experiment *MAS-492* achieved a MASE metric of **2.36**.

Below is a simplified pseudo code for the algorithm which resets hidden states during training.

1. time_series_counter = 0

2. time_series_number_of_batches_list = [TS.length() // batch size for each TS in time-series]

3. start_training_loop()

4. after each batch: if current batch number == time_seires_number_of_batches_list[time_series_cour then - reset_hidden_state()

5. time_series_counter += 1

6.8 Hybrid method - CNN-AE and LSTM

The Convolutional Autoencoder and LSTM model consists of two individual models that are conjoined. Consisting of two models, the autoencoder and the LSTM model.

6.8.1 Convolutional Autoencoder

As part of the hybrid convolutional autoencoder and LSTM model structure, a convolutional autoencoder is needed. The autoencoder is intended to encode and reconstruct the input values of a time series, before the reconstructed values can be used as input for the LSTM model.

The model is created using 1D convolutional and trans-convolutional layers, encoding the spacial data from the time series.

Model selection

In order to find a well-suited autoencoder design, manual tuning of the model was conducted. Tuning of the model was done incrementally, with different compositions of layer types and sizes. Using layers such as the convolutional 1-dimensional layer, dense layers, MinMaxPooling, BatchNormalization, and different dropouts, an ideal model architecture was tested.

After tuning the autoencoder on the different datasets with both local, global, univariate and multivariate experiments, a shared model design was reached. The convolutional autoencoder consists of an encoder component using 2 convolutional layers. The first layer has a kernel size of 3, with 16 filters, while the second layer has a kernel size of 5, with 32 filters. Similarly, the decoder is comprised of two TransConvolutional layers with kernel sizes 5 and 3, where the first layer has a filter size of 32. The number of filters in the last layer depends on the type of model created. A univariate model has only 1 filter, as only one value of reconstructed per time point, while the multivariate model has 4.

This model architecture ensures the data is well reconstructed, and can be shared across all the different models. All the chosen parameters are shown in Table 8.5 and Table 8.6 in the appendix.

Performance metric

Due to the fact that the only aim for the autoencoder is to construct a recreation of the time series data, it has no need for the same measure of accuracies as the SARIMA model and the LSTM model. This is because it does not perform future predictions, and it is therefore not compared to the predictive models. However, it has its own goals, which it aims to achieve. In order to tune the

autoencoder, a loss function is selected to be used during the training process. Taking inspiration from Zhao et al. [2019], the error metrics MAPE was tested. However, during tuning and testing of the models, the MAPE metric was found to make predictions with extremely high error values. The same goes for the MSE metric. Therefore, based on experimentation and results from tuning, the MAE metric was found to be the best match for the autoencoder.

Local and global models

With the current design of the autoencoder, the aim was to find an autoencoder that works well with both local and global models. Even though the global models are tasked with encoding more data, this was not a clear problem with the design, although there are limitations. With highly correlating data, such as dataset 1, less data is required to be coded within the model because partial data is shared across the different time series. However, this is not the case for non-correlating data, such as with dataset 2. With this, more data is needed to be encoded with the same model as above. Although this impaired the performance of the autoencoder to some degree, testing and tuning of models found that there are minimal difference in performance between the global and local models. Therefore, both due to limitations with time and with the results from model selection and tuning, the same model was selected as a good fit for all model structures. The same autoencoder model architecture is therefore shared across all explored models.

Univariate and multivariate

Same as with the use of local and global models, the use of univariate and multivariate models with the autoencoder served as a challenge. The selected model needed to be able to encode and reconstruct data for both univariate and multivariate data sources.

Although multivariate models require more data to be encoded and reconstructed, the current autoencoder design works well. Due to experimentation with univariate and multivariate models, it is found that the model works well and can be used on all methods regardless of univariate or multivariate model structure.

6.8.2 LSTM

The second part of the hybrid model is the LSTM model. The LSTM model is attached to the end of the convolutional autoencoder, processing the output of the autoencoder as the model input. This is intended to alter the input data to some extent, removing unneeded noise and volatility.

As with the LSTM models described in Section 6.7, a stateful LSTM model is used in the experiments. Therefore, the same limitations and challenges are present.

The stateful LSTM model requires the use of the same batch size for each pass-through, thus creating problems for datasets where the number of inputs does not match a multiple of the batch size. Section 6.7 explains how this problem is resolved, and the hybrid method applies the same approach. Additionally, the Section 6.7 present the approach for working with global and local methods. The same process is applied to the LSTM module in the hybrid model.

6.8.3 Connected model

Unlike the LSTM models Section 6.7 the hybrid model is not specifically tuned for each of the experiments. Instead, the LSTM models found during LSTM tuning are applied in the hybrid model. This is done primarily in order to create a one-to-one comparison between the LSTM and the hybrid model, where the only difference is the addition of the convolutional autoencoder.

6.9 Statistical T-test

The t-test is used as a statistical measure of significance in predictions between different predictive models, Section 2.1.16. By using the error metric values produced by the predictive models, the student t-test is used in order to verify whether or not the predictions made by two different models are significantly different.

The error value of each time series per dataset is extracted after the experiments are executed. These results are then aggregated in lists of error values. A list is created per predictive model. The lists can then be compared using the t-test, testing for a significant difference between two and two lists of error metrics from different models.

The results from the t-test can then be used as a measure to evaluate if two model predictions were made on the same dataset is of significant difference.

Chapter 7

Experiments and Results

This chapter presents the results accomplished through the execution of the experiments defined in chapter Chapter 6.

Initially, the project experiment plan is presented in Section 7.1. This contains the definition of each of the experiments executed. Additionally, this section connects each experiment with its related research question, as well as the expectations of the experiment.

Secondly, the results from the experiments are presented in Section 7.2. Both with tables of the average experiment metrics compared and box plots for results from each of the datasets.

Lastly, additional experiments were completed exploring models on datasets with other abilities. Section 7.3 and Section 7.4 lists the additional experiments conducted, using data decomposition or high noise data.

7.1 Experimental Plan

This section contains the definition of the different experiments conducted on the datasets defined in Section 5.2 applying the methods and models presented in Chapter 6.

7.1.1 Experiment 0 - SARIMA Baseline

This experiment will serve as a measurement for the predictive accuracy of other predictive methods. The SARIMA baseline will be used to compare the other methods applied in later experiments. This experiment will not answer any research questions.

Outline Tune, train and test a seasonal SARIMA method on dataset 1, dataset 2, and dataset 3 using MASE and sMAPE as metrics on a 7-day forecast horizon.

Expectations The SARIMA method is expected to perform better on more seasonally dependent data, while not performing exceedingly well in comparison to other models.

These experiments will serve as the baseline metric for comparison with later experiments.

7.1.2 Experiment 1 - LSTM Baseline

This experiment will serve as a baseline created using a currently state of the art predictive method for E-commerce future prediction, as pressented in Section 4.2. By comparing the results achieved in this experiment with the ones from Section 7.1.1, this experiment serves to answer RQ4 Section 1.3.

Outline Tune, train and test a local univariate LSTM on dataset 1, dataset 2, and dataset 3 using MASE and sMAPE as metrics on a 7 day forecast horizon. The results are compared with the results from the SARIMA baseline. Section 7.1.1

Expectations The univariate local LSTM model is expected to outperforme the SARIMA model on every dataset.

7.1.3 Experiment 2 - LSTM Model structures

This experiment attempt to improve the predictive ability of the LSTM model, using multivariate and global methods. The experiments are designed to create models with the aim of outperforming the baseline SARIMA and Local univariate LSTM models created in Section 7.1.2 and Section 7.1.1. This experiment aims to answer RQ4.1 [Section 1.3], through the exploration of the questions outlined below.

- Will additional information, such as day of the week, month and season help a LSTM to make better predictions?
- Will giving a LSTM model additional data by training a LSTM across multiple datasets improve predictions?

Outline Tune, train and test a local multivariate LSTM, a global univariate LSTM, and a global multivariate LSTM on dataset 1, dataset 2, and dataset 3 using MASE and sMAPE as metrics on a 7-day forecast. Compare the results against the SARIMA and LSTM baseline.

Expectations It is expected that the multivariate models will outperform the univariate models. The additional date-encoding of seasonality should help the NN make more accurate predictions. It should not impair the model accuracy.

Following the works of Montero-Manso and Hyndman [2021] described in Section 3.3.1 the global models are expected to perform, at worst, equal to the local model on all datasets, while at best outperforming the local models.

7.1.4 Experiment 3 - Convolutional Autoencoder LSTM

This experiment focuses on the proposed hybrid Convolutional Autoencoder and LSTM model. As this method has yet to be tested on a commercial dataset, the experiment aims to improve the accuracy of the correlating LSTM models. The hybrid model is explored in each of the experiments ran on the LSTM, including all combinations of Local vs Global models, and univariate vs multivariate models.

This experiment aims to answer RQ5 [Section 1.3] by applying the Hybrid method to the datasets.

Outline Tune, train and test a CNN-AE and LSTM models to compare to the LSTM models created in Section 7.1.2 and Section 7.1.3. Creating a local univariate model, local multivariate model, global univariate model, and global multivariate model. All these models are applied to datasets 1, 2 and 3, using MASE and sMAPE metrics on a 7 day forecast. The results are compared against the SARIMA and LSTM models defined in previous experiments.

Expectations The Convolutional Autoencoder and LSTM model is expected to perform better than the correlating LSTM model due to the high volatility and noise in the available datasets. The multivariate models are expected to perform better than the univariate models. However, the global models are expected to perform well on dataset 1 due to the highly correlating data, while performing worse on datasets 2 and 3.

7.2 Results

This section presents the results acquired from executing the experiments defined in Section 7.1. Results are presented with average values of error metrics per dataset, as well as box-plots of error metrics, and t-test results between the explored models.

All results for each of the experiments can be found in Section 8.8.

7.2.1 Dataset 1

Table 7.1: Average values for all experiment for dataset-1

Experiment	MASE	sMAPE	MASE-7
sarima	1.294	0.239	1.063
local univariate lstm	1.129	0.208	1.017
local multivariate lstm	0.972	0.183	0.922
global univariate lstm	1.107	0.203	1.02
global multivariate lstm	1.091	0.202	1.055
local univariate cnn ae lstm	1.124	0.208	0.947
local multivariate cnn ae lstm	**0.943**	**0.181**	**0.871**
global univariate cnn ae lstm	1.157	0.21	1.058
global multivariate cnn ae lstm	1.082	0.198	1.026

Table 7.2: Student t-test, measuring confidence of significant difference between predictions, comparing LSTM model structures against local univariate LSTM. sMape error - p-value

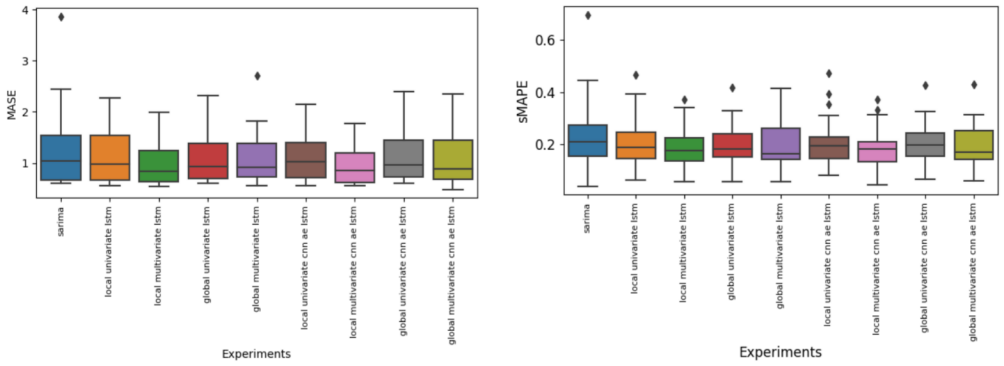
	local_multivariate	global_univariate	global_multivariate
dataset 1	0.039211	0.712811	0.642842
dataset 2	0.228875	0.638309	0.320224
dataset 3	0.139856	0.384286	0.185118

Table 7.3: Student t-test, measuring confidence of significant difference between predictions on the CNN-AE-LSTM and the LSTM for different model structures. sMape error - p-value

	local-univariate	local-multivariate	global-univariate	global-multivariate
dataset 1	0.96549	0.72792	**0.00027**	0.35168
dataset 2	0.14676	**0.01051**	**0.03044**	**0.0103**
dataset 3	0.82601	0.85881	0.15914	**0.02646**

Table 7.1 shows the mean metrics across all the time series in dataset 1. Table 8.53 show the p-values for every LSTM structure compared with the local

Figure 7.1: Boxplot of predictions made on dataset 1



(a) MASE boxplot of predictions made on dataset 1

(b) sMAPE boxplot of predictions made on dataset 1

univariate LSTM. Table 7.3 show the p-values comparing the sMAPE results between the LSTM and the CNN-AE-LSTM for each model structure. The poorest performing model across the board is SARIMA, with a MASE of 1.294, sMAPE of 0.239 and 7-day MASE of 1.063. All the different LSTM structures and hybrid methods outperformed SARIMA, as seen in Figure 7.1b.

The multivariate models outperformed all their retrospective univariate counter partners. The global univariate LSTM outperforms the local univariate on both MASE and sMAPE, though not statistically significant, but this results is not reproduced for the multivariate models, or the CNN-AE-LSTM models.

All the model structures performs slightly better with our proposed CNN-AE-LSTM method on dataset 1, except for the global univariate CNN-AE-LSTM.

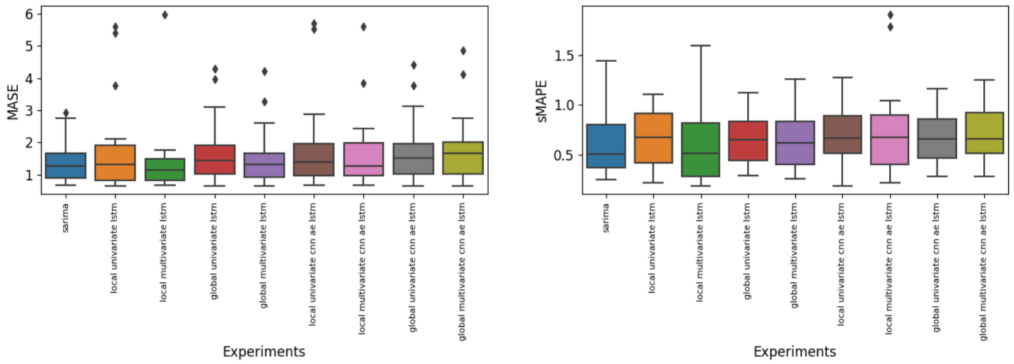
The most consistent models with the least amount of variance are the multivariate models. The local multivariate CNN-AE-LSTM and the local multivariate LSTM were the only two models who beat the naive 7-day prediction. The best performing model is the local multivariate CNN-AE-LSTM.

7.2.2 Dataset 2

Table 7.4: Average values for all experiment for dataset-2

Experiment	MASE	sMAPE	MASE-7
sarima	1.472	0.633	0.707
local univariate lstm	1.765	0.684	0.812
local multivariate lstm	**1.377**	**0.603**	**0.697**
global univariate lstm	1.707	0.662	0.834
global multivariate lstm	1.55	0.642	0.775
local univariate cnn ae lstm	1.827	0.716	0.851
local multivariate cnn ae lstm	1.66	0.743	0.816
global univariate cnn ae lstm	1.734	0.675	0.85
global multivariate cnn ae lstm	1.796	0.703	0.89

Figure 7.2: Boxplot of predictions made on dataset 2



(a) Boxplot of the MASE metrics

(b) Boxplot of the sMAPE metrics

The results from dataset 2 are shown in Table 7.4, and visualized in Figure 7.2. The p-values are in Table 8.53 and Table 7.3.

Compared to dataset 1, dataset 2 proved to be a much more difficult dataset to forecast. Compared to the neural network models, SARIMA performs better, with a mase of 1.472, a 7-day MASE of 0.707 and a smape of 0.633. Some of the results from dataset 1 are reproduced on dataset 2. For example, the global univariate LSTM outperforms the local univariate LSTM. The multivariate

models outperform the univariate models. On dataset 2 however, all the CNN-AE-LSTM methods perform poorly compared to the LSTM models. The best performing model on dataset 2 is local multivariate LSTM with a mase of 1.377 a sMAPAE of 0.603 and 0.697.

7.2.3 Dataset 3

Table 7.5: Average values for all experiment for dataset-3

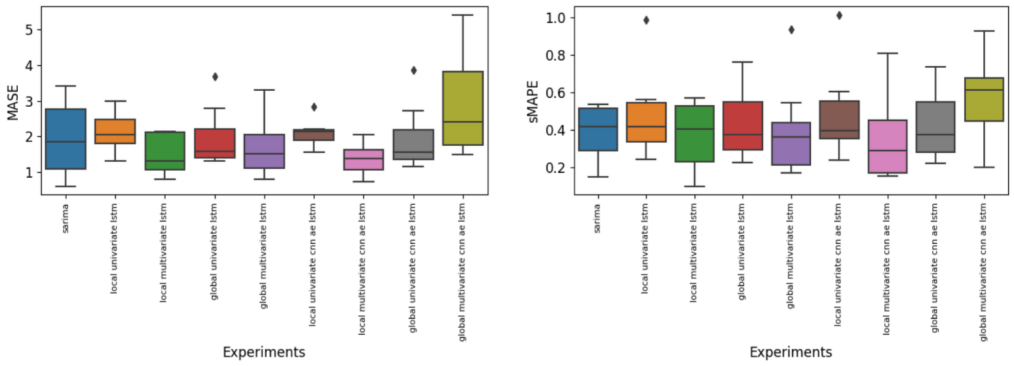
Experiment	MASE	sMAPE	MASE-7
sarima	1.938	0.382	0.981
local univariate lstm	2.135	0.477	1.306
local multivariate lstm	1.495	0.369	0.988
global univariate lstm	1.968	0.425	1.038
global multivariate lstm	1.728	0.399	0.899
local univariate cnn ae lstm	2.106	0.483	1.242
local multivariate cnn ae lstm	**1.352**	**0.354**	**0.765**
global univariate cnn ae lstm	1.94	0.417	1.004
global multivariate cnn ae lstm	2.935	0.566	1.771

The results from dataset 3 is in Table 7.5, and visualized in Figure 7.3. The p-values are in Table 8.53 and Table 7.3. On dataset 3 SARIMA, again, outperforms the purely local univariate LSTM baseline. Many of the same patterns repeats on dataset 3 as with dataset 1 and 2, multivariate beats univariate, global unviariate beats local univariate. On dataset 3 it seems that the CNN-AE-LSTMs outperform the LSTMs once again.

7.2.4 Model Comparisons Across Datasets

Due to the low sample size of the available datasets, proving the statistical significance of model performance proves to be a difficult task. However, aggregating the results across the 3 datasets will increase the sample size of the predictions. This will, in turn, improve the basis for doing statistical tests on the model predictions. The results are shown in Figure 7.4

Figure 7.3: Boxplot of predictions made on the seasonal dataset 3



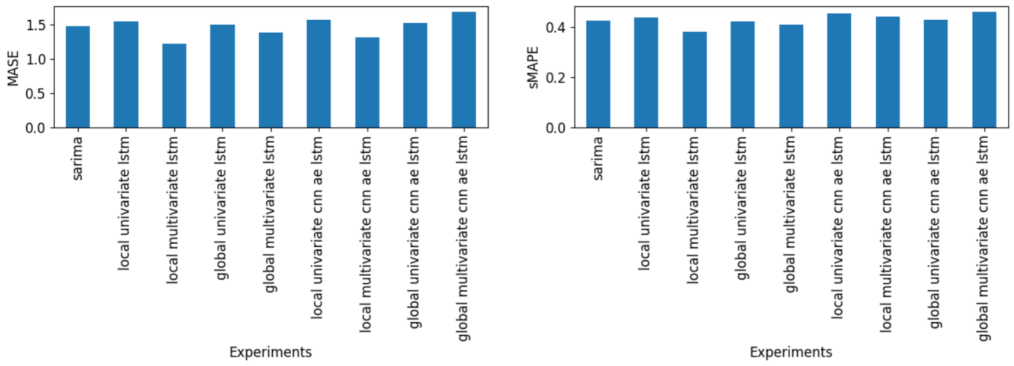
(a) MASE boxplot. The local multivariate CNN-AE-LSTM performs best, while the global multivariate CNN-AE-LSTM performs worst.

(b) sMAPE boxplot. The local multivariate CNN-AE-LSTM performs best, but with a higher variance than on MASE. The global multivariate CNN-AE-LSTM performs worst.

Table 7.6: Average values for all experiment for all-datasets

Experiment	MASE	sMAPE	MASE-7
sarima	1.481	0.425	0.905
local univariate lstm	1.552	0.438	0.977
local multivariate lstm	**1.221**	**0.382**	0.842
global univariate lstm	1.497	0.423	0.946
global multivariate lstm	1.382	0.41	0.912
local univariate cnn ae lstm	1.573	0.454	0.954
local multivariate cnn ae lstm	1.311	0.442	**0.832**
global univariate cnn ae lstm	1.524	0.429	0.962
global multivariate cnn ae lstm	1.686	0.46	1.095

Figure 7.4: Barplot comparing average model perfomance across all datasets



(a) Average MASE metrics across all datasets

(b) Average sMAPE metrics across all datasets

Table 7.7: Student t-test, measuring confidence of significant difference between LSTM models all dataset, statistic value. sMape error. l-u = local univariate, l-m= local multivariate, g-u = global univariate etc. - p-value

	l-u	l-m	g-u	g-m
l-u	nan	**0.03918**	0.36293	0.10352
l-m	**0.03918**	nan	0.10993	0.23882
g-u	0.36293	0.10993	nan	0.19305
g-m	0.10352	0.23882	0.19305	nan

LSTM models

The global univariate LSTM model improves prediction error over the local univariate model by 2.4% with the sMAPE metric. Additionally, applying the t-test implies that this difference in metric is not significant, with a p-value of 0.712. However, in order to measure the difference in the models we compare the models across all of the datasets. By aggregating the prediction metrics, as shown in Table 7.6, we achieve an improvement of 4.43% instead. Additionally, the p-value from the statistical t-test has a significant decrease in value to 0.362, as shown in Table 8.54. Although this is still not enough for the t-test to indicate a statistically significant difference, we are able to see a clear trend indicating that the global univariate model is an improvement compared to the local univariate

model.

The local multivariate LSTM shows a big improvement over the local univariate LSTM. On average, the performance of the model reduces predictive error by 13.74% using the sMAPE error metric. Additionally, the p-value of the t-test shows a value of 0.039, thus implying a significant difference in predictions.

The global multivariate LSTM model improves predictive error by 4.2% over the local univariate LSTM model, and 3.1% over the global univariate model. In addition to outperforming the local univariate baseline, the global multivariate model also outperformed the global univariate model. However, the t-tests for the model are all above the threshold of 0.05 compared to both models. The global multivariate model is not able to outperform the local multivariate model, with a predictive error of 7.33% worse than the local multivariate model.

7.2.5 Hybrid model compare results

The hybrid methods can be compared with the LSTM baseline methods. The experiment results used in this section comes from results tables Table 7.1, Table 7.4, Table 7.5, as well as the tables for t-test values, Table 7.3.

Local univariate

The local univariate LSTM and the local univariate convolutional autoencoder and LSTM are one of the model-configurations that offer rather similar results across datasets. While the hybrid model performance is more or less equal, predictions made on datasets 2 and 3 vary a bit more. The basic local univariate LSTM model performs 4.6% and 1.25% better on average than the local univariate CNN-AE-LSTM model. However, in addition to these similarities in average performance, the T-test conducted on the experiments reveals the same outcome. The t-test Resultsults shown in Table 7.3 infer that the predictions made by the two local univariate models are from the same group, and there is no significant difference in their distribution.

Global univariate

The global univariate CNN-AE-LSTM model is, on average, an improvement over the local univariate CNN-AE-LSTM model. The model outperforms the local univariate model on datasets 2 and 3, but not on dataset 1. However, the global model performs better than the local model on average over all datasets with a 5.5% lower sMAPE error metric. The global univaraite CNN-AE-LSTM model is not able to outperforme the global univariate LSTM model. The global univariate LSTM model generally performed better than the hybrid model. Evaluating the difference with the t-test, it is also clear that the difference is significant.

Although the average metrics performance between the hybrid model and the LSTM model on dataset 1 is only a little under 4%, there is still quite a difference. By evaluating the metrics of each prediction in the dataset, as well as using the significance test results from the t-test, it is clear that the global univariate LSTM model is significantly better than the global univariate convolutional autoencoder and LSTM for dataset 1. For dataset 2 the difference is not as prominent as with dataset 1, but also here, there is a significant difference in predictions. Only with dataset 3, where the hybrid model performs better than the LSTM model on average as a global univariate model, the t-test signals that there are no significant differences between the two predictions. However, it is unclear if this is due to the dataset or if this has to do with the low sample size compared to dataset 1 and dataset 2.

Local multivariate

The local multivariate CNN-AE-LSTM model serves a slight improvement to predictions for datasets 1 and 3, although there are no clear improvements to predictions. On average, dataset 1 is improved with 1.1% and dataset 3 is improved with 1.4% for the sMAPE error metric. The t-test applied to the predictions supports this assumption as there are no significant differences between the predictions made with the local multivariate model.

However, the results are quite different for dataset 2. The CNN-AE-LSTM model increases the sMAPE error metric with 23% over the local multivariate LSTM model, thus decreasing performance significantly. This is also supported by the t-test conducted, implying a significant difference in prediction sMAPE errors.

Global multivariate

On dataset 1 the global multivariate hybrid model performs on average 2% better than the global multivariate LSTM model. However, there is not a significant difference. This is supported by the results from the t-test, which implies there is no significance in the difference in predictions. While there was a significant difference between the models using a global univariate model, the difference in performance was not high.

Unfortunately, the same does not apply to dataset 2 and dataset 3. While the hybrid model, on average, perform about 9.5% worse on dataset 2, it also performs about 41.9% worse on dataset 3. Applying the student t-test to the predictions also supports the notion of the global multivariate model decreasing performance for the convolutional autoencoder and LSTM. Both predictions for dataset 2 and dataset 3 are well within the confidence interval of the t-test, thus implying a significant difference between the predictions for datasets 2 and 3.

7.3 Additional Experiments Plan

After analyzing the initial results from our defined experiments we formed some hypotheses, which led to some additional experiments. These experiments are described in this section.

7.3.1 Experiment 4: CNN-AE-LSTM on Noisy datasets

The CNN-AE-LSTM performed best on dataset 1, which we know has the most noise. To confirm our hypothesis that the CNN-AE-LSTM performs better than a LSTM on the noisy dataset we add three new datasets, one set consists of noisy time series, the other consists of time series with low noise. In fear that the noisiest dataset would be too noisy to give any meaningfule predictions we added the third dataset, which has above-average noise, but not as much as the noisiest data set.

Choosing the datasets

To make the two datasets we looked at the standard deviation. Firstly we scaled down outliers which could skew the standard deviation of a series to be higher. For this we used the same technique as described in Section 6.5.7.

Then, for each time series in the whole dataset we did these steps

1. Scale down the values to have a mean of 0 using standardization.
2. Take the difference of $t_n - t_{n-1}$
3. if the standard deviation of the remaining series is above $max\_threshold$ Add to max list
4. if the standard deviation of the remaining series is below the $min\_threshold$ Add to min list

The noisiest dataset had standard deviation above $max\_threshold = 1.3$. The dataset with the least noise had a standard deviation below $min\_threshold = 0.4$. The above average noisy dataset had a standard deviation above $ok\_threshold = 0.8$ but below the maximum threshold of 1.3. From the remaining list we randomly picked 7 time series. Examples from each dataset is shown in Figure 7.5, and the full list of categories are presented in Table 7.8, Table 7.9, and Table 7.10.

Running the experiments

The only two model structures we use are local univariate LSTM and local univariate CNN-AE-LSTM. We follow the same procedure as previously described

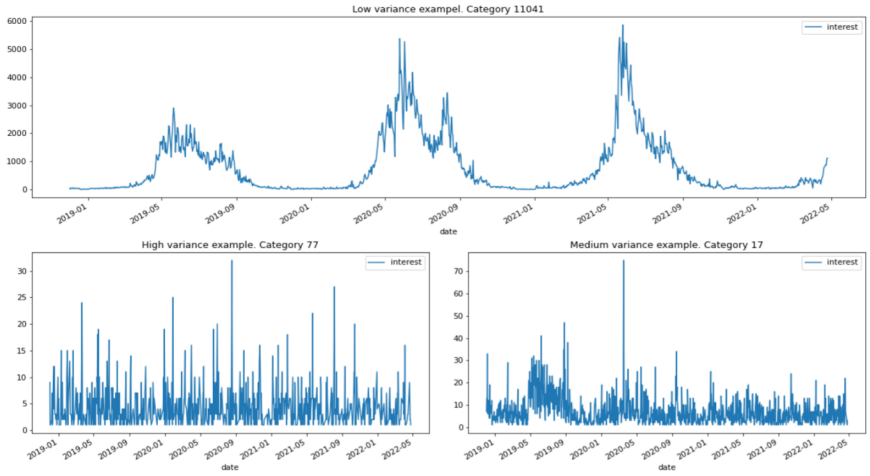


Figure 7.5: Illustration of time series from the low variance dataset (top), high variance (bottom left), and the medium variance dataset (bottom right).

in Chapter 6, except we do not tune the hyperparameters. Instead, we use a hyperparameter-set from a previously tuned experiment and use those for all the models. This is done because of time constraints.

Experiment Plan

Outline Train and test a local univariate LSTM, and a local univariate CNN-AE-LSTM on a noisy dataset, a low noise dataset, and a medium noisy dataset, using MASE and sMAPE as metrics on a 7-day forecast. Compare the results against each other.

Expectations We expect the CNN-AE-LSTM to outperform the LSTM on the noisy datasets. We also expect the LSTM to have the best performance on the low noise datasets.

7.3.2 Experiment 5: Differencing on seasonal dataset

The literature on RNN's ability to model seasonality was conflicting [Section 6.5.6]. Our initial results showed that the LSTM suffered on datasets with strong yearly seasonality. We want to test if removing trends and seasonality can improve forecasts.

Table 7.8: Categories chosen for the highl variance dataset

Category ID	Name (Norwegian)
77	Harddiskkassett
79	Viftestyring
11217	Brannmur (Firewall)
11334	Lamineringsmaskin
11992	Lommelerke
12265	Bilderamme
12511	Sårrens

Table 7.9: Categories chosen for the low variance dataset

Category ID	Name (Norwegian)
10053	Sykkel
11037	Grill
11041	Gressklipper
11048	Hekksaks
11456	Robotgressklipper
11817	Gresstrimmer og kantklipper
13323	Brodder

Choosing Datasets

We chose dataset 3 as the main dataset and dataset 1 as a control dataset. Dataset 3 consists of series with strong seasonality, while dataset 1 has a relatively low seasonality.

Removing Trend and Seasonality

To remove the trend and seasonality from the time series we use differencing transformation, which is a method of transforming a time series, which removes its temporal dependence [Rob J Hyndman, 2014, p. 215]. STL decomposition was tested as well, but it proved to be give worse results than with differencing. Differencing is performed by subtracting the previous observation from the current observation as shown in Equation (7.1) where z is the transformed differenced series, and y is the real observations.

$$\mathrm{z}(t) = y(t_n) - \mathrm{y}(t_{n-1}) \tag{7.1}$$

Table 7.10: Categories chosen for the ok variance dataset

Category ID	Name (Norwegian)
16	Diverse lydkort
17	Media
28	Skanner
34	Videokamera
40	Optiske enheter
44	Vifter
45	Mobilkabler
48	Blåtann-adapter

The transformation can be removed with Equation (7.2).

$$\hat{y}(t) = z(t_n) + y(t_{n-1}) \tag{7.2}$$

Running the experiments

We would follow the same experimental procedure as previously described in Chapter 6, except for section Section 6.5.6 which we replace with differencing described above.

Experiment Plan

Outline Tune, train and test a local univariate LSTM with differencing on dataset 1 and 3, using MASE and sMAPE as metrics on a 7-day forecast. Compare the results against previously ran experiment without differencing.

Expectations We expect the LSTM with differencing to perform better on dataset 3. We do not know how differencing will affect the results on dataset 1.

7.4 Additional Experiments Results

Table 7.11: Average values for all experiment for dataset-variance

Experiment	MASE	sMAPE	MASE-7
dataset-high-variance-lstm-local-univariate	0.645	0.864	0.796
dataset-high-variance-cnn-ae-lstm-local-univariate	**0.634**	0.832	0.782
ok-variance-lstm-local-univariate	0.788	0.502	0.757
ok-variance-local-univariate-cnn-ae-lstm	0.788	0.501	**0.747**
low-variance-lstm-local-univariate	3.74	**0.421**	0.898
low-variance-cnn-ae-lstm-local-univariate	5.072	0.641	1.082

Table 7.12: Student t-test, measuring confidence of significant difference between predictions, statistic value. sMape error - p-value

	low-variance-lstm-local-univariate
High variance	0.14734
OK variance	0.73271
Low variance	**0.01043**

Table 7.13: Student t-test, measuring confidence of significant difference between predictions, statistic value. MASE error - p-value

	low-variance-lstm-local-univariate
High variance	0.38205
OK variance	0.9745
Low variance	0.05382

The results of our two additional experiments are shown in Table 7.11. And the p-values are presented in Table 7.13, and Table 7.12. The results are also visualized in Figure 7.6. The table shows the average metrics for all the time series in the dataset described in the name. For example: *dataset-high-variance-lstm-local-univariate* row show the average results for the local univariate LSTM on the high variance dataset.

7.4.1 CNN-AE-LSTM on variance

The CNN-AE-LSTM got MASE of 0.645 and an sMAPE of 0.864 on the high variance dataset. The CNN-AE-LSTM (*dataset-high-variance-cnn-ae-lstm-local-unviariate*) outperforms the LSTM on the same dataset with an sMAPE of 0.832 and a mase of 0.634, which is an sMAPE reduction of 3.7% and a relativily low p-value of 0.147.

The CNN-AE-LSTM performs much worse than the LSTM on the low variance dataset, whit a sMAPE of 0.641 versus 0.421, and a MASE of 5.072 versus 3.740. This is a 52% decreased sMAPE perfomance and a low p-value of 0.01. The MASE score is 26.3% worse with a p-value of 0.053.

On the medium variance dataset the CNN-AE-LSTM performs marginally better than the LSTM with a sMAPE of 0.501 versus 0.502. Both models score the same MASE of 0.788, the p-value of 0.732 is too high to say anything of value.

7.4.2 Differencing

Table 7.14: Average values for all experiment for dataset-diff

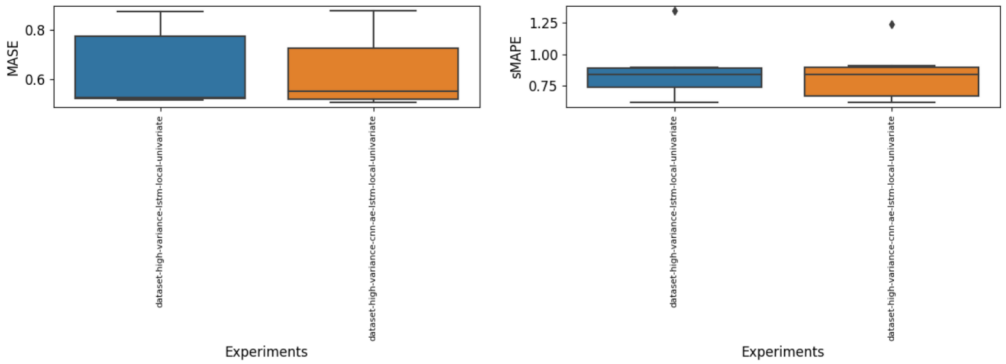
Experiment	MASE	sMAPE	MASE-7
local univariate lstm dataset 1	**1.129**	**0.208**	1.017
local univariate lstm dataset 3	2.207	0.488	1.345
local univariate lstm dataset 1 diff	1.341	0.24	0.98
local univariate lstm dataset 3 diff	1.866	0.367	**0.916**

Table 7.15: Student t-test, measuring confidence of significant difference between predictions, statistic value. MASE error - p-value

	local-univariate	local-univariate
Differencing dataset	0.18392	0.50322

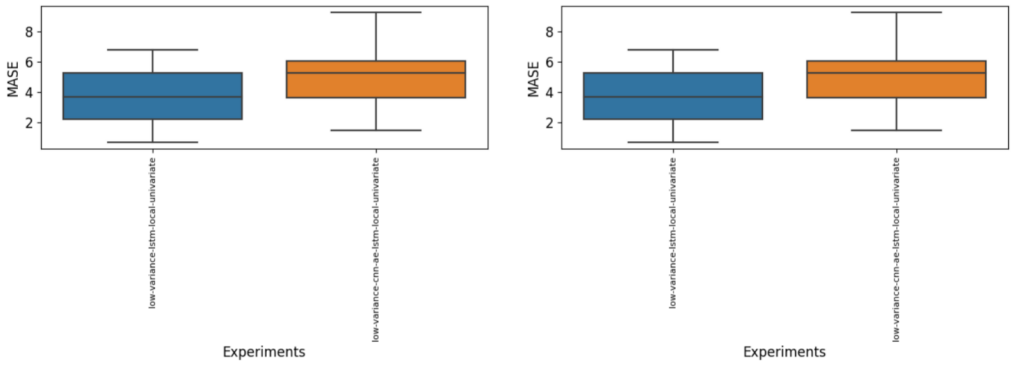
Comparing the local univariate LSTM with differencing done as a pre-processing step on dataset 3 in Table 7.14, with the same model type, without differencing, the original local univariate LSTM model without differencing scored a MASE of 2.207, a sMAPE of 0.488, and a 7-day MASE of 1.345. The model with differencing scored a MASE of 1.866, a sMAPE of 0.367 and a 7-day MASE of 0.916. That is an improvement of 15.45%, 24.80%, and 31.90% respectively;

Figure 7.6: Boxplot of predictions made on the high variance, and the low variance dataset, comparing CNN-AE-LSTM against LSTM



(a) MASE metrics from the high variance dataset. The CNN-AE-LSTM show improvements.

(b) sMAPE metrics from the high variance dataset. The CNN-AE-LSTM show im-



(c) MASE metrics from the low variance dataset

(d) sMAPE metrics from low variance dataset

however, none of these improvements are not of statistical significance as the sMAPE p-value is as high as 0.503, as shown in Table 8.44.

We can, however, with greater confidence confirm that differencing on dataset 1 will decrease the results by 15.38%, with a p-value of 0.183.

Chapter 8

Discussion and Conclusion

This section will cover the discussion and conclusion of this thesis. Section 8.1 presents the discussion regarding the proposed framework, selected model structures, datasets and data analysis, and the results from the experiments. Section 8.2 presents the scientific contributions done in this thesis, followed by Section 8.3 which concludes the thesis. Lastly, Section 8.4 discuss threats agains validity, before Section 8.5 lists future work to be done on the problem space and the presented framework.

8.1 Discussion

This section addresses the underlying discussion creating the basis for the selected model framework proposed in Chapter 4 and Chapter 6, and the related experiments and results defined in Chapter 7. The discussion contains the reasoning behind method selection and design and the motivation behind the selection of error metrics and data processing. Experimental results are addressed in order to discuss the selected methods and to answer the research question and goal defined in Section 1.3.

8.1.1 Datasets Characteristics

The results reveal a lot of information about the characteristics of the different datasets. The sMAPE is our best indicator for how well the prediction graph fits the target graph. The 1-day MASE gives us an indicator of how well our predictions are compared to the naive prediction. It is worth noting how well the naive prediction differs for each time series. In a random-walk the naive prediction will be the best possible prediction. This is also true for time series

with a high autocorrelation coefficient. The same is true for the 7-day MASE. A time series with a strong weekly seasonality will give a higher MASE compared to a time series with a low weekly seasonality. Therefore comparing MASEs from different datasets can be misleading without knowledge of the underlying characteristics of these series.

Looking at the results we can make some educated guesses about these time series characteristics. We get the best forecast fits from dataset 1, which has a mean sMAPE of 0.2034. Second comes dataset 3 with a sMAPE of 0.4362. The hardest dataset to forecast was dataset 2 with a sMAPE of 0.697. Looking at the 1-day MASE we can expect the mean autocorrelation coefficient for dataset 3 to be the highest among the three, with a MASE of 2.009. Calculating the autocorrelation for all the time series in the datasets and taking the average of the results confirms our hypothesis. The results are shown in Table 8.1. Dataset 3 has the highest autocorrelation of all the datasets with a mean of 0.93. There is also a relatively strong correlation between the 1-day MASE results from the local univariate LSTM method and the time series autocorrelation.

Table 8.1: Mean autocorrelation and the correlation of the MASE results from Local Univariate LSTM and the autocorrelation.

Dataset	Mean Autocorrelation	Autocorrelation correlation with MASE
1	0.753	0.229
2	0.775	0.295
3	0.930	0.304

In general, using sMAPE as metric, dataset 2 seems to be the most difficult dataset to forecast. This makes sense because the dataset consists of many vastly different time series with few apparent patterns.

8.1.2 Modeling seasonality

Our empirical results indicate that LSTMs have trouble modeling yearly seasonality. The dataset with the least yearly seasonality is dataset 1. This is the only dataset where the local univariate LSTM outperformed SARIMA. However, feeding the LSTM with additional data such as day of the week, month, and season, it became significantly better on datasets with a strong seasonal component. Our findings do not contradict the findings of Hewamalage et al. [2021] regarding NNs ability to model seasonality [Section 6.5.6] because all their datasets have a much higher seasonal frequency. Their data show clear seasonal patterns in a plot with 50 time steps. In contrast, we have yearly seasonality so we have to plot 365+ time steps to be able to see a seasonal pattern.

It seems LSTMs do not have long enough memory to model seasonal patterns with a 365 timestep wavelength. This hypothesis is supported by the findings of Zhao et al. [2020], which concludes that RNNs and LSTMs do not have long memory from a time series perspective. But they do not give a definite answer for how long a LSTM remembers.

This explains why SARIMA outperforms the local univariate LSTM on dataset 2 and 3, as both includes of time series with a strong seasonal dependence. Experiment 5 [Section 7.3.2] where we removed the seasonality from dataset 3 confirms this as well, but the experiment had too few samples to prove anything statistically. When we remove trend and seasonality by differencing the training data, we have to reincorporate this difference on our forecasted values before we can compare them with our test targets. This is done iteratively by Equation (7.2) where the first $z(t_1)$ is our first predicted value and our first observed $y(t_{n-1})$ value is the last value we have before the forecast horizon. This process is based on only one real observation, which means that we accumulate an error for each iteration, which can hurt the forecast accuracy. This probably explains the poor results on dataset 1 when differencing is performed in Experiment 5.

Regarding RQ4 [1.3], we can conclude that on our dataset, when both SARIMA and LSTM are given the exact same information, SARIMA will outperform LSTM on time series with a strong 365-day seasonality because of the spatial memory limitations of LSTMs. SARIMA is a statistical method that does not rely on memory. It requires the seasonal component for each time series to be known. When a time series does not have a strong seasonal dependence or this dependence is removed beforehand, the LSTM performs best.

Regarding RQ4.1 [1.3] our empirical results show that adding date stamp features to a univariate series can significantly improve forecasting accuracy, especially on time series with a strong seasonal dependence, and where this seasonal component has a long wavelength.

8.1.3 Global versus Local models

Comparing the local univariate against the global univariate model, the global model has an sMAPE performance increase of 2.4% on dataset 1, 3.2% on dataset 2 and 4.9% on dataset 3. The sample sizes of each dataset are relatively low, so the results are not statistically significant. However, when we increase the sample size by comparing the same models across all the datasets, we see the same trend and we get a lower p-value. This show a promising trend. Surprisingly, the performance increase does not seem to correlate with how homogeneous the time series in the dataset are to each other. Dataset 1, the most homogeneous set of them, has the least performance increase. These results support Montero-Manso and Hyndman [2021] preposition that global models can improve forecasting ac-

curacy, even if the strong assumption that the same process generates the time series. And that a global algorithm can also express each forecast that a local algorithm can express.

The same performance increase is not to be found in the multivariate models. The local multivariate models are good at capturing the trend and seasonality of a given product category. This does not seem to translate well to a global model. One might expect that global models should be able to learn seasonality across time series if the series contains the same seasonality. The dataset that suffers the most from making a multivariate local model global is dataset 3 with a sMAPE perfomance loss of -37%. Dataset 1 and dataset 2 got -10.38% and -6.47% loss respectively. One explanation for this might be that even though all categories in dataset 3 are popular during the winter, and peaks around the same months, their seasonality is not enough in sync. For example, *"Vintersko"* and *"Vinterjakke"* has their most giant peaks around October when the weather starts becoming cold. While *"Langrennski"* and *"Skisko"* peaks around January, when the snow starts falling. This will seem like conflicting information for an NN that can not differentiate between which category it is looking at, hurting the NNs modeling capability. However, the value of additional information is not to be overlooked, because our results show that a multivariate global LSTM will outperform a univariate global LSTM in all scenarios. And the global model is a lot more scalable, which can make it is an attractive model structure choice, even if it is not the most accurate model structure.

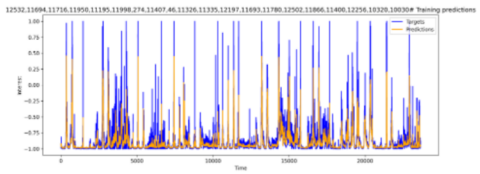
Regarding RQ4.1 [1.3] our empirical results show that training a global univariate model across multiple time series will simplify the problem, and improve forecasting accuracy, even if the underlying time series have very different characteristics. However, this is not true for multivariate models with additional date stamp features. We can assume that global multivariate models with date features will work if all the set time series have the same seasonal patterns. Local multivariate models will give the most accurate forecast, but global multivariate models scale a lot better as the number of time series to forecast increases.

8.1.4 Normalization versus Standardisation

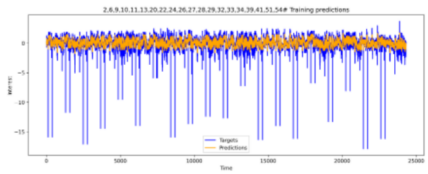
Our initial assumption was to use normalization to scale the data between a fixed range between -1 and 1 because we did not know if the data would follow a Guassian distribution. After testing both normalization and standardization, standardization proved to perform significantly better. The problem with scaling all values between a fixed range is that huge outliers use up a lot of the range available, which will result in most of the data having very small differences in values between them. This can affect the NNs ability to differentiate between the observations.

Figure 8.1 show two different time series scaled with both techniques

Figure 8.1: Effects of different scaling techniques on a dataset with huge outliers.



(a) A time series with huge outliers scaled using normalization. Most values are squeezed between -1 and -0.75.



(b) A time series scaled using standardization. Most values are centered between -2 and 2.

8.1.5 Convolutional Autoencoder with LSTM

After doing experiments with the LSTM and the CNN-AE and LSTM model on the three datasets used in this thesis, the results from the predictions are presented in Section 7.2.5.

Experiments conducted on the different datasets result in predictions that can vary vastly, and some that are quite similar between the models.

The t-test and average metrics attained through the experimentation presented in Section 7.2.5 infer that the hybrid model CNN-AE-LSTM is not able to improve performance over the LSTM model on a local univariate model.

One reason behind this lack of difference in predictions might originate in the selected data. The task of the convolutional autoencoder is to encode and reconstruct the input data of the model, removing noise and other unneeded information in the process. Lack of data could serve as an issue in this case. As each of the time series used in the local univariate model only consists of around three years of data, or 1300 data points. There is a possibility that there is not enough similar data for the autoencoder to recognize similarities in data so that it can then remove the extra noise. Additionally, the amount of noise contained in the data is not known. If the datasets contain little noise, the autoencoder is more likely to remove relevant data, thus impairing the performance of the LSTM model. Due to the implications that the lack of noise in the data potentially could infer from the predictions, additional experiments are done on data with high noise and low noise. We will come back to these experiments later in this discussion.

However, as is seen from the predictions made by the two models, it is clear that the convolutional autoencoder and lstm does not significantly improve pre-

dictions compared to the LSTM model.

While the thesis Zhao et al. [2019] only focuses on applying local univariate models, we wished to explore the expansion of the use of the hybrid model with other configurations. These include the use of multivariate models and global models for each dataset. Exploring the use of such variations to the Convolutional autoencoder and LSTM, we hoped to uncover cases where the model performance is increased over the standard local univariate model.

Local multivariate models

After the use of global models, a configuration of multivariate models was attempted. Unlike the global models, the amount of data is increased not through the addition of other time series to the model, but by decomposing the information within each time series. By decomposing information such as day of the week, or season, the amount of data supplied to the models is increased.

Using the hybrid convolutional autoencoder and LSTM model should result in both improvements and degradation in performance.

Initially, the performance of the model is likely to suffer due to the fact that the model is required to encode and reconstruct additional input data using the same model structure. The model might encounter problems recreating the data using the same model as for the univariate model while attempting to recreate additional data per data point. If the autoencoder then is limited by the number of data entries on top of this, the model performance could suffer.

However, the same reasoning might also help improve the hybrid model's performance in some situations. While more data would need to be encoded and reconstructed, the autoencoder will retain more information regarding the development of trends and spikes through different seasons. While a pattern might repeat over several seasons, there might be additional hidden information in the seasonal information. If a pattern is known to never spike during summer but often spikes through the winter, the autoencoder should be able to differentiate between the different seasons. Thus, the task of the autoencoder would be to retain information about changes in trends also dependent on seasonal data.

However, while this reasoning is based on the autoencoder's ability to encode and differentiate between values based on season, the amount of data would severely limit the model. If there is not enough data available, the autoencoder would not be able to learn of such connections. As discussed in Section 5.1, the data attained from "Prisguiden.no" only include data-points for a little over three years. As with the problems discussed with the local univariate model, the multivariate model would also likely be limited by the amount of data.

This is reflected in the results acquired through testing. The results of these tests are introduced in Section 7.2.5, and while the performance of datasets 1 and 3 are a little better than for the LSTM model, the differences are not significant.

However, the performance of the method on dataset 2 is shown to be significantly worse than the LSTM predictions.

The significant decrease in performance could be due to the lack of data as discussed earlier. With the local univariate model, there is a 4% performance decrease with the hybrid model. Although this is not significant, this could point to the reason behind the poor performance. While the local univariate model and the local multivariate models both suffer from a lack of data with each dataset. Both models perform worse on dataset 2, while the performance on dataset 1 is somewhat better both using univariate and multivariate. This could imply that the selected dataset is less suited for using the autoencoder. This could then again imply that the convolutional autoencoder is heavily influenced by the nature of the data that is used. The implications of the dataset's characteristics are explored further later when the use of noisy data is discussed. However, if this is the case, the lack of data could explain the poor performance of dataset 2. The multivariate model would have the same number of time-steps in each time series but would be required to encode four times the data per time-step. Therefore, the autoencoder is more likely to perform worse in reconstructing the input data, thus contributing to a worse performance by the LSTM component.

Although the local multivariate model is the overall best performing of the convolutional autoencoder and LSTM models, it is outperformed by the LSTM, indicating that the reason for the improvements by the model is entirely contributed by the improved LSTM model.

Global univariate models

After having explored different local model configurations, global models are also tested. As is discussed in Section 8.1.3, global univariate models generally outperform the local models, training on more available data for the neural network. The same is mostly true for the convolutional autoencoder, which performs better with a global univariate configuration than with local univariate models.

As shown in Section 7.2.5, it is clear that with a global univariate configuration, the hybrid model is not able to outperform the LSTM model and often performs worse.

It seems the benefits of a global model for RNNs are not transferable to other types of NNs. LSTMs ability to learn across multiple time series can be attributed to the fundamentals of how the LSTM works. Zhao et al. [2019] discusses previous work that concludes that the LSTMs memory cell is mainly responsible for the performance of the LSTM. When training the LSTM on multiple time series, this memory cell is local to each time series and resets before each new time series are fed through the network. The LSTM weights, on the other hand, are global across all the series. These local and global characteristics are unique to RNNs, and can explain why the Convolutional Autoencoder does not see the same benefits

when trained on multiple time series. In order for the autoencoder to work well on such time series data, there is a prerequisite that the data is connected and that information from one set can be applied to the others. If this is not the case, the use of global models increasing the amount of independent data would only serve to decrease the performance of the autoencoder.

Global Multivariate models

Both multivariate models and global models have been tested with the hybrid convolutional autoencoder and LSTM. The next step is to apply a model with both of these configurations. We, therefore, apply a global multivariate convolutional autoencoder and LSTM, and measure the performance of the model compared to a global multivariate LSTM model.

It is clear that the global multivariate model suffers the same problems that are prevalent in both the local multivariate model and the global univariate model. Like with the global univariate models, the autoencoder is not as well suited as predicted to encode multiple independent time series. However, performance degradation is also influenced by using a multivariate model. The autoencoder needs to encode and reconstruct more data per data-point, and with the same autoencoder structure as well as a lack of data, the autoencoder will likely perform worse than with the use of a univariate model.

These assumptions are reflected in the results acquired from running experiments on the global multivariate models. While the performance of dataset 1 is more or less equal between the baseline global multivariate LSTM model and the hybrid model, the same is not the case for dataset 2 and dataset 3. The results can be found in Section 7.2.5.

Using the local multivariate model, the hybrid model performed somewhat better than the LSTM model for dataset 1. It appears that the same applies to the global multivariate model. Although it has not improved the performance of the hybrid model significantly it seems that the use of a multivariate model has counteracted the adverse effects of the global model on dataset 1.

Performance degradation for dataset 2 was expected due to the poor results both with the global univariate model and the local multivariate model. Both of these configurations resulted in worse predictions on dataset 2. Dataset 3 on the other hand, while expected, is not as easily explained. While the performance was not particularly good on either of the previous models, it would appear that the combination of the multivariate model and global model caused degradation in performance. However, the vast difference in performance could also be attributed to the low sample size of the dataset. While this is not the only reason, it might be a contributing factor to the large difference.

Either way, it is clear that the convolutional autoencoder is not well suited for use with global multivariate models, as it appears to vastly decrease the

performance of the predictions.

CNN-AE-LSTM on high noise datasets

As we discussed when talking about the performance of the convolutional autoencoder and LSTM on datasets 1, 2, and 3, we made the assumption that the characteristics of the used dataset could have severe implications on the predictive accuracy of the hybrid model.

In order to test this, additional experiments were defined and run as described in Section 7.3. By applying these experiments using the hybrid model to 3 new datasets with varying degrees of data noise, the assumption was that the degree of noise in the dataset would heavily influence the predictive abilities of the hybrid model.

Due to the design of the hybrid model, applying a convolutional autoencoder to the input data of the model, the data would be altered in accordance with the autoencoder. The assumption is that when applying the autoencoder to a dataset with a low amount of noise, the autoencoder would be more likely to remove important information on which the LSTM part of the model is dependent on. On the other hand, with a higher level of noise in the dataset, the model is more likely to reduce the noisy values, contributing to input data that is easier for the LSTM model to interpret.

The results from the experiment substantiate the hypothesis above. Using the first noise dataset (low noise), it is apparent that the CNN-AE-LSTM greatly decreases the performance of the predictions. With a 52% decrease in performance, it is clear that the model is not well suited to make predictions on such low noise data. This notion is further supported by the t-test value signaling that the difference in predictions is significantly different.

However, applying the hybrid model to the ok-noise dataset is more successful. Although it does not improve the accuracy and reduce the error metric of the predictions, it performs about as well as the LSTM model. There is only a 0.2% difference in predictive error, and the t-test supports the claim that it does not impair the performance of the LSTM model.

By running experiments on a dataset with low noise and ok noise, it is clear that the dataset used with the hybrid model has a strong influence on performance. Low noise is shown to reduce the performance of the hybrid model, while the problem does not occur in data with medium/ok noise. Despite this, it is not clear if the amount of noise in the data only can impair the performance, or if a more suitable amount of noise could possibly improve the performance. In order to explore if this is the case we have also explored a dataset with a high level of noise as defined in Section 7.3. Unlike the datasets with low or ok noise, the high noise dataset proved to have more success with the CNN-AE-LSTM. On average, the hybrid model performs a little under 4 percent better than the LSTM

model using the sMAPE error metric. Although the t-test is not able to verify the confidence interval that the predictions are significantly different between the LSTM and the hybrid model, we are able to manually evaluate the predictions made by the two models on the dataset. Although there are some variations in the predictions, we are able to see a general trend of the hybrid model increasing the performance over the LSTM model on the high variance.

Analyzing the results from the model predictions on the three datasets with high, low, and ok variance, it is made clear that the dataset used in connection with the CNN-AE-LSTM and LSTM has a strong influence on the predictive ability of the model. Higher variance is easier for the model to predict and recreate meaningfully, while data with lower noise serve as more of a problem with the autoencoder.

Considering the level of variance/noise in the dataset has such an impact on the hybrid model, it is clear that other factors could also have a significant implication on the use of the hybrid model. The dataset used by Zhao et al. [2019] could contain other characteristics that make the hybrid model well suited for predictions, something that might not be available in the dataset from "Prisguiden.no".

One theory is that the more clearly seasonal data used in the paper by Zhao et al. [2019] strongly influence the results. The traffic flow dataset illustrated in the paper as one of the datasets used had a clear and repetitive seasonal pattern that occurs much more frequently than in our available dataset. This could be an advantage for the autoencoder as it would have more similar data to work with and, therefore, more easily reconstruct data without data noise that is generally not contained in the rest of the dataset.

Additionally, the presence of noise in the dataset is unclear. While the dataset from prisguiden is comprised of multiple datasets with varying degrees of noise, the datasets available to Zhao et al. [2019] could be more suited for the use of the autoencoder due to the presence of noise in the dataset.

While these assumptions are difficult to test and prove in the given state of afears, this is an important point for further investigation and work with the hybrid model in order to test and verify its viability of use in other contexts or problems. While we are not able to achieve the same results as Zhao et al. [2019], there are many possible reasons for this. Additionally, while we are not able to prove a statistically significant improvement to forecasting throug the use of the hybrid model, we are able to see a shared trend of improved results using the method. We might not be able to recreate the same results as Zhao et al. [2019] with such large improvements to the error metric values, this is largely dependent on different factors such as the available dataset.

8.2 Contributions

Explore the use of a hybrid convolutional autoencoder and LSTM on E-commerce data from "Prisguiden.no". The datasets from "Prisguiden.no" have not previously been explored through the use of time series predictions, and all experimentation done on the dataset is new research. In addition to testing the hybrid model on the dataset, we also expanded on the experimentation with the hybrid model. The model is based on the work done in Zhao et al. [2019] where the model type is proposed on a time series problem. However, the model tested here is only a local univariate model, and no experimentation data was found on the use of either global or multivariate model variations. Therefore, we increase the experimental domain on which the proposed hybrid model is applied and tested, in an effort to evaluate the model's predictive ability with new use-cases.

8.3 Conclusion

The goal of this thesis is to forecast future trends using a convolutional autoencoder and LSTM model, explore different model structures, and create some guidelines for forecasting the dataset supplied by "Prisguiden.no". We reach this goal by answering the research questions RQ4, RQ4.1 and RQ5 from Section 1.4.

Answering RQ4, we can conclude that a univariate LSTM outperforms SARIMA unless a strong yearly seasonality is present. However, the LSTM short memory can be compensated for by adding additional features to the LSTM containing temporal information. If additional temporal information is unavailable then removing seasonalities with data pre-processing techniques such as differencing, might improve accuracy on highly seasonal datasets, but will hurt accuracy on non-seasonal datasets.

Further, regarding RQ 4.1, we can conclude that both a multivariate LSTM, and a global LSTM improves forecasting accuracy in every scenario. However, the combination of the two model structures is not preferred if accuracy is the prime objective. The local multivariate LSTM, with features that describe the date, is the preferred method for attaining the lowest predictive error. However, a global multivariate model can be a good alternative if scalability is essential.

Finally, regarding RQ5 our results, although not statistically significant, with a p-value of 0.147, show that a hybrid Convolutional Autoencoder can improve a LSTM on time series with a lot of noise. However, we can confirm that the hybrid model can severely hurt accuracy if the datasets are not noisy enough. Thus, the performance increases are not enough to justify using a hybrid CNN-AE-LSTM over a LSTM on our dataset.

8.4 Threats Against Validity

All processes are subject to compromises. Some of these choices might compromise the validity of the results. This section summarizes all threats we could find that might affect the results or make the results difficult to reproduce.

8.4.1 Small sample sizes

We had to limit ourselves to a relatively small sample size. For most experiments we have three datasets, each dataset contains 20 samples, 20, samples, and 8 samples. Summed up that's 48 samples in total. The high variance in the results, combined with a small sample size makes it difficult to prove our results with a sufficiently high statistical confidence. We can however show promising trends, and the fact that the same trends in our result hold, while the p-value decreases, when we increase the sample size by comparing across datasets is promising.

8.4.2 Generelizability the results

All of the time series past are relevant for the future. "Prisguiden.no" logs new data each day, so when we executed the final experiments on 26.04.2020 we used all the available data up to that point. Since the domain of E-commerce is volatile and is constantly changing, we cannot prove that our results hold for any other points in time. For example, can the buying patterns for our chosen categories change a lot a couple of years from now. Therefore we try in this report to focus on the underlying characteristics of the time series, rather than the specific product categories.

8.4.3 Fixed Batch size

As described in Section 6.7 we had to set our batch size to a fixed number. This is not something the papers we base our method on are doing. However, none explains in detail if they are using a stateful LSTM or how they are implementing it. If we had solved the problem of batch sizes and stateful LSTM, we might have found that we did not use the optimal batch size for our problem.

8.4.4 Reseting LSTM States

In section Section 6.7 we describe how we resets our LSTM states when training a global model. We describe how we can in some cases reset the states too late. This is because a batch might contain two different time series. This can hurt our results.

8.5 Future Work

The work done in this thesis investigates the use of the LSTM and convolutional autoencoder and LSTM on the E-commerce domain of "Prisguiden.no". However, the research done in this thesis is not entirely practically applicable. This comes from the fact that only a small sub-set of the entirety of the available data has been applied in the research done. This was a limitation applied due to the scope of the thesis and time limitations that adhere to the research done in a master's thesis. Therefore, applying the models explored in this thesis to a real-world application remains a valid future task.

Additionally, while the priority of this thesis was to validate and explore the use of a convolutional autoencoder and LSTM in a time series prediction setting, other models might also fit well with the task of making such predictions. The Facebook model Prometheus, XGB, and LightGBM, are all valid models that can also be tested on the dataset to evaluate their usability.

The research conducted in this thesis focuses primarily on forecasting a period of 7 days. While a 7-day prediction does have value, there might be some merits to increasing the forecasting window. By increasing the forecasting window from 1 week to 30 days (a month), or 60 days (2 months), the use case for these predictions would empower "Prisguiden.no" as discussed Chapter 8. While a 30 day prediction period was briefly tested, the 7-day prediction has been prioritized in this research. Increasing the forecasting window is, therefore, a possible point of entry for further research conducted on the dataset from "Prisguiden.no".

Although the experiments done in this thesis are done on several unique datasets, more work can be done with the experimentation of hybrid model used on datasets with different characteristics. While dataset 1 and 2 focus on corrolation between time series, this was selected primarily for the experimentation with local vs global models. The additional testing defined in Section 7.3 explores the use of the convolutional autoencoder and LSTM on datasets with higher levels of noise and lower levels of noise. The same concept can be applied to create datasets with other characteristics. By doing this, the hybrid model performance can be explored more in-depth using different characteristics of data. It is expected that the type of data selected for experimentation is highly influential on the performance of the model, and further experimentation with the model could therefore be warranted.

E-commerce is a domain that is influenced a lot by external factors, such as holidays and seasonal sales such as black friday. Expanding the number of features from external sources is a domain worth exploring for future work.

Bibliography

Anders Kofod-Petersen (2018). How to do a Structured Literature Review in computer science. Technical report.

Bandara, K., Bergmeir, C., and Smyl, S. (2017). Forecasting Across Time Series Databases using Recurrent Neural Networks on Groups of Similar Series: A Clustering Approach. Expert Syst. Appl., 140.

Bandara, K., Shi, P., Bergmeir, C., Hewamalage, H., Tran, Q., and Seaman, B. (2019). Sales demand forecast in E-commerce using a long short-term memory neural network methodology. Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), 11955 LNCS:462–474.

Ben Taieb, S., Bontempi, G., Atiya, A. F., and Sorjamaa, A. (2011). A review and comparison of strategies for multi-step ahead time series forecasting based on the NN5 forecasting competition. Expert Syst. Appl., 39(8):7067–7083.

Bharadi, V. A. and Alegavi, S. S. (2021). Analysis of Long Short Term Memory (LSTM) Networks in the Stateful and Stateless Mode for COVID-19 Impact Prediction. pages 167–190.

Bowen, T., Zhe, Z., and Yulin, Z. (2020). Forecasting method of e-commerce cargo sales based on ARIMA-BP model*. Proc. 2020 IEEE Int. Conf. Artif. Intell. Comput. Appl. ICAICA 2020, pages 133–136.

Cerqueira, V., Torgo, L., and Soares, C. (2019). Machine Learning vs Statistical Methods for Time Series Forecasting: Size Matters.

Chu, C. W. and Zhang, G. P. (2003). A comparative study of linear and nonlinear models for aggregate retail sales forecasting. Int. J. Prod. Econ., 86(3):217–231.

Géron, A. (2017). Hands-On Machine Learning with Scikit-Learn and TensorFlow. O'Reilly Media, 1 edition.

Hale-Evans, R. (1998). Oreilly @ Internet Medicine Show. CMAJ, 159(12):1–1511.

Hewamalage, H., Bergmeir, C., and Bandara, K. (2021). Recurrent Neural Networks for Time Series Forecasting: Current status and future directions. Int. J. Forecast., 37(1):388–427.

Hyndman, R. J. and Koehler, A. B. (2006). Another look at measures of forecast accuracy. Int. J. Forecast., 22(4):679–688.

Kenton, W. (2020). Time Series Definition.

Khan, Z. A., Hussain, T., Ullah, A., Rho, S., Lee, M., and Baik, S. W. (2020). Towards Efficient Electricity Forecasting in Residential and Commercial Buildings: A Novel Hybrid CNN with a LSTM-AE based Framework. Sensors 2020, Vol. 20, Page 1399, 20(5):1399.

Laptev, N., Yosinski, J., Erran Li, L., and Smyl, S. (2017). Time-series Extreme Event Forecasting with Neural Networks at Uber. Int. Conf. Mach. Learn., pages 1–5.

Makridakis, S., Spiliotis, E., and Assimakopoulos, V. (2018). Statistical and Machine Learning forecasting methods: Concerns and ways forward. PLoS One, 13(3):e0194889.

Montero-Manso, P. and Hyndman, R. J. (2021). Principles and algorithms for forecasting groups of time series: Locality and globality. Int. J. Forecast., 37(4):1632–1653.

Ouyang, Z., Ravier, P., and Jabloun, M. (2021). STL Decomposition of Time Series Can Benefit Forecasting Done by Statistical Methods but Not by Machine Learning Ones. page 42.

Rabanser, S., Januschowski, T., Flunkert, V., Salinas, D., and Gasthaus, J. (2020). The Effectiveness of Discretization in Forecasting: An Empirical Study on Neural Time Series Models.

Ramos, P., Santos, N., and Rebelo, R. (2015). Performance of state space and ARIMA models for consumer retail sales forecasting. Robot. Comput. Integr. Manuf., 34:151–163.

Rob J Hyndman (2014). Forecasting: Forecasting: Principles & Practice. Number September.

Russel, S. and Norvig, P. (2012). Artificial Intelligence: A Modern Approach. Knowl. Eng. Rev.

Sen, R., Yu, H. F., and Dhillon, I. (2019). Think Globally, Act Locally: A Deep Neural Network Approach to High-Dimensional Time Series Forecasting. Adv. Neural Inf. Process. Syst., 32.

Sharda, R. and Patil, R. B. (1992). Connectionist approach to time series prediction: an empirical test. J. Intell. Manuf., 3(5):317–323.

Sindre Sivertsen, S. K. (2021a). Decision Matrix. https://northern-leech-f32.notion.site/6e1d2fe02ed0428e99c841216abc78af?v=79637ea21f724a9dbe18fc04ad476bd3.

Sindre Sivertsen, S. K. (2021b). Jupyter Notebook Market insight data exploration. https://github.com/sjsivert/Masteroppgave/blob/master/notebooks/market_insight_data_explorati

Sindre Sivertsen, S. K. (2021c). Search Term Table. https://northern-leech-f32.notion.site/ae2904fd23114215b4faf040c291c6d6?v=47e2bb2fbb384e83a85afe8fb4fad30a.

Sindre Sivertsen, S. K. (2021d). Structured Literature Review Database. https://northern-leech-f32.notion.site/1a5ea8cd5ce64e0bbbdbbe8dae75e295?v=40c41c5c48184bcf8cf8d25f3e61fa7d.

Sindre Sivertsen, S. K. (2021e). Structured Literature Review, Final paper cut-off. https://northern-leech-f32.notion.site/1a5ea8cd5ce64e0bbbdbbe8dae75e295?v=40c41c5c48184bcf8cf8d25f3e61fa7d.

Sindre Sivertsen, S. K. (2022). sjsivert/Masteroppgave: Master project for Sindre J. I. Sivertsen and Sander Kilen. https://github.com/sjsivert/Masteroppgave/.

Sivertsen, S. and Kilen, S. (2021). Exploring Hybrid Methods for E- commerce Time-Series Forecasting. Technical report, NTNU.

Smyl, S. (2020). A hybrid method of exponential smoothing and recurrent neural networks for time series forecasting. Int. J. Forecast., 36(1):75–85.

Snoek, J., Larochelle, H., and Adams, R. P. (2012). Practical Bayesian Optimization of Machine Learning Algorithms. Adv. Neural Inf. Process. Syst., 4:2951–2959.

Utlaut, T. L. (2008). Introduction to Time Series Analysis and Forecasting, volume 40. Wiley-Interscience.

Van Hoa, T., Anh, D. T., and Hieu, D. N. (2021). Foreign Exchange Rate Forecasting using Autoencoder and LSTM Networks. ACM Int. Conf. Proceeding Ser., pages 22–28.

Weng, T., Liu, W., and Xiao, J. (2020). Supply chain sales forecasting based on lightGBM and LSTM combination model. Ind. Manag. Data Syst., 120(2):265–279.

Yu, H. F., Rao, N., and Dhillon, I. S. (2016). Temporal regularized matrix factorization for high-dimensional time series prediction. Adv. Neural Inf. Process. Syst., pages 847–855.

Zhang, G. P. and Qi, M. (2005). Neural network forecasting for seasonal and trend time series. Eur. J. Oper. Res., 160(2):501–514.

Zhang, X. and You, J. (2020). A Gated Dilated Causal Convolution Based Encoder-Decoder for Network Traffic Forecasting. IEEE Access, 8:6087–6097.

Zhao, J., Huang, F., Lv, J., Duan, Y., Qin, Z., Li, G., and Tian, G. (2020). Do RNN and LSTM have Long Memory?

Zhao, X., Han, X., Su, W., and Yan, Z. (2019). Time series prediction method based on Convolutional Autoencoder and LSTM. Proc. - 2019 Chinese Autom. Congr. CAC 2019, pages 5790–5793.

Ziegel, E. R., Box, G., Jenkins, G., and Reinsel, G. (1995). Time Series Analysis, Forecasting, and Control, volume 37. John Wiley & Sons.

Zunic, E., Korjenic, K., Kerim, H., and Donko, D. (2020). Application of Facebook's Prophet Algorithm for Successful Sales Forecasting Based on Real-world Data. Int. J. Comput. Sci. Inf. Technol., 12(2):23–36.

Appendices

8.6 Experiment Framework: Example config

```
---
data:
  data_path: './datasets/raw/market_insights_overview_all_2022_02_14.csv'
  #categories_path:  './datasets/raw/solr_categories_all_16_09_2021.csv'
  categories_path: './datasets/raw/solr_categories_all_2022_02_14.csv'

logger:
  log_level: 'INFO'
  log_file: './log-file.log' # Currently does nothing

use_gpu_if_available: false # Some experience indicate that GPU is slower

experiment:
  tags:
    - market_insights
  save_sources_to_use: # Which sources the experiment should be saved to
    - 'disk'
    - 'neptune'

  checkpoint_save_location: './models/0_current_model_checkpoints/'
  log_model_every_n_epoch: 10

  # Possible metrics: MSE, MAE, MSE; MAE
  error_metrics:
    - 'MAE'
    - 'MASE'
    - 'MSE'
    - 'SMAPE'
    - 'MAPE'

  save_source:
    disk:
      model_save_location: './models/'

    neptune:
      project_id: 'sjsivertandsanderkk/Masteroppgave'
      # Set api token with env variable NEPTUNE_API_TOKEN
      #api_token: ${NEPTUNE_API_TOKEN}


model:
  # Model types: 'validation_model'. 'local_univariate_arima', 'local_univariate_lstm', 'local_cn
  # global
  model_type: 'local_univariate_arima'
  rng_seed: 42
  validation_model:
    placeholder: 0
  univariate_lstm:
    hyperparameter_tuning_range:
      hidden_size: [ 2, 100]
      number_of_layers: [ 1, 4]
      dropout: [ 0.0, 0.4 ]
      #optimizer_name: ['Adam', 'RMSprop']
      optimizer_name: ['RMSprop', 'Adam']
      #optimizer_name: ['Adam']
```

```
    learning_rate: [ 1e-7, 1e-2 ]
    number_of_epochs: [5, 40]
    batch_size: [32, 32]
    input_window_size: 10
    output_window_size: 7 # must be equal output_window_size
    multi_variable_nr: 4 # must be equal to number of variables used in multi variable (1
    number_of_features: 4 # must be equal to number of features in data
    number_of_trials: 200 # Number of tuning trials to run. The more the better.
    #time_to_tune_in_minutes: 6000  # Time to tune in minutes, If both number of trials
    stateful_lstm: true

  global_model:
    parameters_for_all_models:
      # If int <= 1 then the testing set will be equal to output_window_size
      input_window_size: 10
      output_window_size: 7
      multi_variable_nr: 4
      batch_size: 32
      number_of_epochs: 22
      stateful_lstm: yes
      should_shuffle_batches: no
      optimizer_name: Adam
      learning_rate: 0.00043177648728211254
      hidden_layer_size: 50
      dropout: 7.057973795771e-06
      number_of_features: 4
      number_of_layers: 1
    datasets:
      - 12256
      - 10320
      - 10030

  local_model:
    common_parameters_for_all_models:
      # If int <= 1 then the testing set will be equal to output_window_size
       input_window_size: 10
       output_window_size: 7
       multi_variable_nr: 1
       batch_size: 32
       number_of_epochs: 15
       stateful_lstm: yes
       should_shuffle_batches: no
       optimizer_name: RMSprop

    model_structure:
      - time_series_id: 12532
        learning_rate: 0.00013756
        hidden_layer_size: 94
        dropout: 0.21199
        number_of_features: 1
        number_of_layers: 1
      - time_series_id: 11694
        learning_rate: 0.00013756
        hidden_layer_size: 94
        dropout: 0.21199
        number_of_features: 1
        number_of_layers: 1

local_univariate_cnn_ae:
  common_parameters_for_all_models:
    input_window_size: 14
    output_window_size: 7
    multi_variable_nr: 1
    batch_size: 1
    number_of_epochs: 51
    optimizer_name: 'adam'
    loss: "mse"
    should_shuffle_batches: True

  model_structure:
    - time_series_id: 12322
      learning_rate: 0.003
      encoder:
        - layer: "Conv1d"
          filters: 8
          kernel_size: 3
          activation: "relu"
        - layer: "MaxPool"
          size: 2
          padding: "valid"
        - layer: "Conv1d"
          filters: 16
```

```
            kernel_size: 3
        decoder:
          - layer: "Conv1d"
            kernel_size: 5
            filters: 8
            activation: "relu"
          - layer: "Conv1d"
            kernel_size: 3
            filters: 4
            activation: "relu"
          - layer: "Conv1d"
            kernel_size: 3
            filters: 1

local_univariate_cnn_ae_lstm:
  common_parameters_for_all_models:
    should_shuffle_batches: True
    optimizer_name: 'Adam'
    loss: "mae"
    batch_size: 10
    epochs: 20
    number_of_epochs: 20
    input_window_size: 10
    output_window_size: 7
    multi_variable_nr: 1
    lstm-shared:
      input_window_size: 10
      output_window_size: 7
      multi_variable_nr: 1
      epochs: 26

  model_structure:
    - time_series_id: 12322
      lstm:
        optimizer_name: 'RMSprop'
        stateful_lstm: True
        loss: "mae"
        learning_rate: 7.88e-05
        hidden_layer_size: 14
        dropout: 0.132
        number_of_features: 1
        number_of_layers: 3
      ae:
        optimizer_name: 'Adam'
        loss: "mse"
        learning_rate: 0.003
        epochs: 20
      encoder:
        - layer: "Conv1d"
          filters: 8
          kernel_size: 3
          activation: "relu"
        - layer: "MaxPool"
          size: 2
          padding: "valid"
        - layer: "Conv1d"
          filters: 16
          kernel_size: 3
      decoder:
        - layer: "Conv1d"
          kernel_size: 5
          filters: 8
          activation: "relu"
        - layer: "Conv1d"
          kernel_size: 3
          filters: 4
          activation: "relu"
        - layer: "Conv1d"
          kernel_size: 3
          filters: 1


local_univariate_arima:
  forecast_window_size: 7
  steps_to_predict: 7
  multi_step_forecast: true # alternative is recursive single step
  auto_arima: true
  seasonal: true
  # Ranges used for autotuning if --tune parameter is set
  hyperparameter_tuning_range:
    p: [1, 3]
    d: [1, 3]
```

```
    q: [1, 3]
    P: [0, 3]
    D: [0, 3]
    Q: [0, 3]
    s: [12, 12]

# metric_to_use_when_tuning: 'MASE'
metric_to_use_when_tuning: 'MAE'

model_structure:
    - time_series_id: 12322
      hyperparameters:
        p: 0
        d: 1
        q: 0
        P: 5
        D: 1
        Q: 0
        s: 4
    - time_series_id: 11428
      hyperparameters:
        p: 2
        d: 1
        q: 0
        P: 5
        D: 1
        Q: 0
        s: 4
...
```

8.7 Model parameters

8.7.1 SARIMA model

Table 8.2: Auto-arima parameters

Parameter	value
start_p	1
d	1
start_q	1
max_p	7
max_d	7
max_q	7
start_P	0
D	None
start_Q	0
max_P	5
max_D	5
max_Q	5
max_order	5
m	12
seasonal	True
stationary	False
information_criterion	'bic'
alpha	0.05
test	'kpss'
seasonal_test	'ocsb'
stepwise	True
n_jobs	1
start_params	None
trend	None
method	'lbfgs'
maxiter	50
offset_test_args	None
seasonal_test_args	None
suppress_warnings	True
error_action	'warn'
trace	True
random	False
random_state	40
n_fits	50
return_valid_fits	False
out_of_sample_size	0
scoring	'mae'
scoring_args	None
with_intercept	'auto'
sarimax_kwargs	None

8.7.2 LSTM

Table 8.3: LSTM cell parameters. Values with a T value was tuned

Parameter	value
units,	T
activation	”tanh”
recurrent_activation	”sigmoid”
use_bias	True
kernel_initializer	”glorot_uniform”
recurrent_initializer	”orthogonal”
bias_initializer	”zeros”
unit_forget_bias	True
kernel_regularizer	None
recurrent_regularizer	None
bias_regularizer	None
activity_regularizer	None
kernel_constraint	None
recurrent_constraint	None
bias_constraint	None
dropout	T
recurrent_dropout	T
return_sequences	False
return_state	False
go_backwards	False
stateful	True
time_major	False
unroll	False

Table 8.4: LSTM dense parameters

Parameter	value
units	outout_window_size
activation	None
use_bias	True
kernel_initializer	"glorot_uniform"
bias_initializer	"zeros"
kernel_regularizer	None
bias_regularizer	None
activity_regularizer	None
kernel_constraint	None
bias_constraint	None

8.7.3 Autoencoder

Table 8.5: Conv1d cell parameters

Parameter	value
filters	16, 32
kernel_size	3, 5
strides	1
padding	"valid"
data_format	"channels_last"
dilation_rate	1
groups	1
activation	"relu"
use_bias	True
kernel_initializer	"glorot_uniform"
bias_initializer	"zeros"
kernel_regularizer	None
bias_regularizer	None
activity_regularizer	None
kernel_constraint	None
bias_constraint	None

Table 8.6: Conv1DTranspose cell parameters

Parameter	value
filters	32, number_of_features
kernel_size	5, 3
strides	1
padding	"valid"
data_format	"channels_last"
dilation_rate	1
groups	1
activation	"relu", None
use_bias	True
kernel_initializer	"glorot_uniform"
bias_initializer	"zeros"
kernel_regularizer	None
bias_regularizer	None
activity_regularizer	None
kernel_constraint	None
bias_constraint	None

8.8 All results

8.8.1 Dataset 1 - Experiment results

Table 8.7: Metrics from experiment, dataset-1, sarima

Category ID	MASE	sMAPE	MASE-7
2	1.892	0.237	3.047
6	0.677	0.216	1.213
9	0.963	0.165	0.837
10	**0.61**	0.15	1.121
11	3.869	0.696	1.348
13	0.613	0.089	1.099
20	0.652	0.352	0.584
22	0.657	0.141	0.82
24	1.471	0.114	1.334
26	0.975	0.175	**0.377**
27	2.44	0.275	0.881
28	1.151	0.383	1.143
29	2.086	0.194	1.283
32	1.169	0.207	0.598
33	1.032	0.157	0.844
34	0.759	0.263	0.694
39	1.447	0.446	1.009
41	0.646	0.274	1.374
51	1.742	0.212	1.167
54	1.035	**0.039**	0.482

Table 8.8: Metrics from experiment, dataset-1, local univariate lstm

Category ID	MASE	sMAPE	MASE-7
2	1.144	0.151	1.805
6	0.748	0.244	1.346
9	0.597	0.097	0.529
10	0.659	0.161	1.21
11	0.901	0.225	0.64
13	**0.553**	0.081	0.985
20	0.637	0.346	0.576
22	0.845	0.178	1.046
24	1.743	0.133	1.496
26	1.049	0.186	**0.407**
27	1.661	0.178	0.599
28	1.5	0.466	1.489
29	2.124	0.192	1.211
32	1.154	0.204	0.593
33	1.704	0.238	1.397
34	0.725	0.25	0.637
39	1.314	0.393	0.92
41	0.674	0.286	1.436
51	0.568	**0.064**	0.95
54	2.277	0.088	1.062

Table 8.9: Metrics from experiment, dataset-1, global univariate lstm

Category ID	MASE	sMAPE	MASE-7
2	2.321	0.279	3.601
6	0.679	0.216	1.234
9	0.91	0.154	0.706
10	0.633	0.156	1.148
11	0.858	0.215	0.6
13	0.607	0.088	1.085
20	**0.605**	0.33	0.545
22	0.714	0.153	0.888
24	1.784	0.141	1.63
26	1.24	0.219	**0.481**
27	1.404	0.15	0.525
28	0.789	0.286	0.726
29	2.037	0.189	1.226
32	0.963	0.174	0.485
33	0.96	0.146	0.802
34	0.661	0.227	0.603
39	1.37	0.416	0.957
41	0.709	0.295	1.519
51	1.382	0.163	0.937
54	1.512	**0.058**	0.702

Table 8.10: Metrics from experiment, dataset-1, local multivariate lstm

Category ID	MASE	sMAPE	MASE-7
2	1.473	0.185	2.307
6	0.541	0.172	0.97
9	**0.539**	0.088	0.5
10	0.626	0.154	1.116
11	0.867	0.216	0.614
13	0.555	0.081	0.986
20	0.631	0.341	0.564
22	0.917	0.189	1.181
24	1.209	0.092	1.093
26	0.851	0.153	**0.326**
27	1.99	0.218	0.717
28	0.745	0.273	1.027
29	1.636	0.151	1.048
32	0.813	0.15	0.426
33	1.23	0.179	0.977
34	0.74	0.256	0.671
39	1.251	0.371	0.874
41	0.549	0.239	1.184
51	0.819	0.096	1.193
54	1.449	**0.056**	0.667

Table 8.11: Metrics from experiment, dataset-1, global multivariate lstm

Category ID	MASE	sMAPE	MASE-7
2	2.706	0.317	4.309
6	0.749	0.242	1.334
9	0.628	0.103	0.59
10	**0.55**	0.136	1.024
11	0.94	0.236	0.65
13	0.603	0.087	1.069
20	0.593	0.321	0.528
22	0.73	0.156	0.914
24	1.796	0.143	1.672
26	0.889	0.16	**0.34**
27	1.43	0.152	0.515
28	0.883	0.309	0.91
29	1.825	0.168	1.159
32	1.061	0.189	0.52
33	1.037	0.156	0.842
34	0.708	0.247	0.646
39	1.368	0.415	0.955
41	0.763	0.311	1.649
51	1.139	0.131	0.818
54	1.419	**0.055**	0.646

Table 8.12: Metrics from experiment, dataset-1, local univariate cnn ae lstm

Category ID	MASE	sMAPE	MASE-7
2	1.176	0.159	1.897
6	0.693	0.221	1.242
9	**0.56**	0.091	0.491
10	0.587	0.145	1.074
11	0.854	0.213	0.608
13	0.578	0.084	1.047
20	0.648	0.352	0.577
22	0.867	0.182	1.062
24	1.54	0.118	1.278
26	0.964	0.172	**0.373**
27	1.356	0.144	0.49
28	1.523	0.472	1.52
29	2.155	0.197	1.361
32	1.08	0.193	0.563
33	1.324	0.193	0.908
34	0.715	0.247	0.636
39	1.312	0.393	0.918
41	0.722	0.309	1.535
51	1.7	0.201	0.382
54	2.121	**0.082**	0.986

Table 8.13: Metrics from experiment, dataset-1, global univariate cnn ae lstm

Category ID	MASE	sMAPE	MASE-7
2	2.392	0.286	3.809
6	0.719	0.23	1.309
9	0.965	0.165	0.781
10	0.636	0.156	1.122
11	0.899	0.225	0.633
13	**0.597**	0.086	1.058
20	0.598	0.325	0.537
22	0.732	0.157	0.899
24	1.926	0.154	1.746
26	1.258	0.223	**0.473**
27	1.451	0.155	0.549
28	0.872	0.309	0.812
29	2.121	0.198	1.268
32	1.099	0.195	0.548
33	0.961	0.146	0.789
34	0.664	0.226	0.606
39	1.393	0.425	0.971
41	0.728	0.302	1.556
51	1.434	0.17	0.936
54	1.703	**0.065**	0.759

Table 8.14: Metrics from experiment, dataset-1, local multivariate cnn ae lstm

Category ID	MASE	sMAPE	MASE-7
2	1.411	0.18	2.329
6	0.574	0.182	1.031
9	**0.55**	0.089	0.492
10	0.622	0.153	1.14
11	0.828	0.205	0.599
13	1.173	0.159	0.901
20	0.618	0.333	0.55
22	0.883	0.183	1.123
24	1.133	0.086	1.031
26	0.616	0.107	**0.194**
27	1.772	0.192	0.626
28	0.608	0.236	0.667
29	1.51	0.14	0.775
32	0.898	0.164	0.476
33	1.28	0.185	1.049
34	0.661	0.224	0.611
39	1.253	0.371	0.876
41	0.713	0.312	1.509
51	0.574	0.065	0.821
54	1.187	**0.045**	0.629

Table 8.15: Metrics from experiment, dataset-1, global multivariate cnn ae lstm

Category ID	MASE	sMAPE	MASE-7
2	2.351	0.283	3.964
6	0.743	0.239	1.309
9	0.684	0.113	0.606
10	0.588	0.145	1.093
11	0.89	0.224	0.633
13	**0.485**	0.071	0.867
20	0.572	0.312	0.51
22	0.669	0.143	0.837
24	2.06	0.167	1.889
26	0.889	0.159	**0.343**
27	1.675	0.181	0.581
28	0.819	0.293	0.876
29	1.768	0.162	1.099
32	0.949	0.172	0.488
33	0.938	0.143	0.78
34	0.698	0.243	0.642
39	1.403	0.428	0.974
41	0.69	0.287	1.51
51	1.212	0.141	0.822
54	1.566	**0.06**	0.693

8.8.2 Dataset 2 - Experiment results

Table 8.16: Metrics from experiment, dataset-2, sarima

Category ID	MASE	sMAPE	MASE-7
12532	1.217	0.512	0.681
11694	0.89	0.386	0.547
11716	0.988	1.083	0.889
11950	1.33	0.749	0.456
11195	2.075	0.259	**0.377**
11998	2.63	0.978	0.751
274	0.754	0.323	0.432
11407	2.933	0.787	0.77
46	**0.676**	0.738	0.947
11326	0.866	0.351	0.752
11335	1.386	0.496	0.926
12197	1.463	0.459	0.704
11693	1.184	**0.248**	0.781
11780	0.783	1.268	0.522
12502	1.509	0.6	0.918
11866	0.855	0.853	0.786
11400	2.537	0.491	0.751
12256	1.373	0.258	0.642
10320	2.758	1.443	1.028
10030	1.223	0.382	0.476

Table 8.17: Metrics from experiment, dataset-2, local univariate lstm

Category ID	MASE	sMAPE	MASE-7
12532	2.017	1.085	1.1
11694	0.841	0.36	0.522
11716	0.683	1.031	0.662
11950	0.845	0.554	0.5
11195	5.398	1.063	0.937
11998	2.102	0.878	0.691
274	1.766	1.082	1.007
11407	0.684	0.329	**0.179**
46	**0.656**	0.695	0.914
11326	1.273	0.57	1.048
11335	1.077	0.349	0.571
12197	1.885	0.648	1.012
11693	1.099	**0.222**	0.661
11780	0.684	1.107	0.665
12502	1.421	0.553	0.868
11866	0.727	0.846	0.669
11400	5.595	0.849	1.643
12256	1.414	0.253	0.658
10320	3.773	0.759	1.401
10030	1.359	0.439	0.528

Table 8.18: Metrics from experiment, dataset-2, global univariate lstm

Category ID	MASE	sMAPE	MASE-7
12532	1.686	0.819	0.939
11694	0.705	0.306	0.44
11716	0.913	1.111	0.884
11950	1.043	0.653	0.574
11195	4.302	0.751	0.776
11998	2.258	0.901	0.789
274	1.646	0.969	0.939
11407	3.971	0.922	1.024
46	**0.651**	0.665	0.913
11326	1.154	0.5	0.997
11335	0.944	0.298	0.648
12197	1.837	0.625	0.95
11693	1.395	**0.294**	0.904
11780	1.501	1.121	1.434
12502	1.608	0.656	0.984
11866	0.779	0.812	0.721
11400	3.099	0.587	0.924
12256	2.086	0.413	0.872
10320	1.176	0.376	**0.431**
10030	1.394	0.452	0.546

Table 8.19: Metrics from experiment, dataset-2, local multivariate lstm

Category ID	MASE	sMAPE	MASE-7
12532	1.518	0.708	0.739
11694	0.905	0.392	0.565
11716	0.701	1.059	0.679
11950	1.568	0.806	0.965
11195	1.406	**0.189**	0.251
11998	1.769	0.825	0.54
274	1.471	0.816	0.83
11407	0.987	0.563	**0.191**
46	**0.658**	0.692	0.922
11326	0.763	0.299	0.621
11335	0.771	0.247	0.504
12197	1.492	0.471	0.926
11693	1.074	0.223	0.646
11780	0.87	1.596	0.848
12502	0.692	0.222	0.45
11866	0.839	1.034	0.769
11400	5.984	0.861	1.792
12256	1.228	0.218	0.626
10320	1.538	0.422	0.574
10030	1.301	0.423	0.504

Table 8.20: Metrics from experiment, dataset-2, global multivariate lstm

Category ID	MASE	sMAPE	MASE-7
12532	1.665	0.808	0.912
11694	0.83	0.359	0.52
11716	0.785	1.045	0.776
11950	0.948	0.611	0.528
11195	4.222	0.735	0.733
11998	2.218	0.916	0.813
274	1.659	0.979	0.93
11407	2.607	0.726	0.715
46	**0.656**	0.678	0.925
11326	1.094	0.467	0.923
11335	0.961	0.305	0.663
12197	1.328	0.407	0.882
11693	1.256	**0.26**	0.852
11780	0.865	1.259	0.832
12502	1.564	0.631	0.982
11866	0.832	0.925	0.755
11400	3.266	0.608	1.027
12256	1.557	0.287	0.722
10320	1.288	0.379	**0.467**
10030	1.395	0.453	0.546

Table 8.21: Metrics from experiment, dataset-2, local univariate cnn ae lstm

Category ID	MASE	sMAPE	MASE-7
12532	1.92	1.004	1.05
11694	0.837	0.36	**0.516**
11716	0.711	1.108	0.691
11950	1.032	0.65	0.597
11195	5.523	1.103	0.966
11998	1.972	0.852	0.665
274	1.919	1.25	1.097
11407	2.026	0.628	0.561
46	**0.664**	0.707	0.923
11326	1.223	0.538	1.037
11335	1.015	0.33	0.618
12197	1.96	0.685	1.089
11693	0.957	**0.19**	0.643
11780	0.966	1.274	0.939
12502	1.458	0.572	0.891
11866	0.72	0.815	0.663
11400	5.695	0.857	1.681
12256	1.744	0.324	0.807
10320	2.875	0.64	1.069
10030	1.331	0.431	0.516

Table 8.22: Metrics from experiment, dataset-2, global univariate cnn ae lstm

Category ID	MASE	sMAPE	MASE-7
12532	1.716	0.842	0.956
11694	0.72	0.309	**0.442**
11716	0.901	1.105	0.878
11950	1.046	0.654	0.571
11195	4.417	0.78	0.796
11998	2.261	0.905	0.784
274	1.672	0.994	0.953
11407	3.767	0.9	0.97
46	**0.651**	0.668	0.914
11326	1.196	0.523	1.035
11335	0.941	0.297	0.637
12197	1.887	0.649	0.953
11693	1.353	**0.283**	0.895
11780	1.697	1.163	1.626
12502	1.634	0.671	0.996
11866	0.788	0.821	0.728
11400	3.134	0.591	0.91
12256	2.145	0.428	0.891
10320	1.404	0.472	0.523
10030	1.359	0.439	0.535

Table 8.23: Metrics from experiment, dataset-2, local multivariate cnn ae lstm

Category ID	MASE	sMAPE	MASE-7
12532	1.261	0.55	0.751
11694	0.896	0.389	0.557
11716	0.717	1.041	0.697
11950	2.07	0.912	0.963
11195	3.845	0.639	0.663
11998	1.635	0.816	0.548
274	1.559	0.893	0.887
11407	1.309	1.02	0.528
46	**0.662**	0.709	0.926
11326	0.983	0.409	0.816
11335	0.711	0.239	**0.473**
12197	2.199	0.805	1.154
11693	1.05	**0.219**	0.694
11780	1.055	1.911	1.041
12502	0.881	0.295	0.568
11866	1.08	1.792	1.005
11400	5.612	0.849	1.665
12256	1.966	0.383	0.984
10320	2.424	0.567	0.896
10030	1.281	0.416	0.508

Table 8.24: Metrics from experiment, dataset-2, global multivariate cnn ae lstm

Category ID	MASE	sMAPE	MASE-7
12532	1.83	0.934	1.01
11694	0.823	0.353	**0.495**
11716	0.843	1.076	0.828
11950	1.049	0.656	0.573
11195	4.851	0.896	0.854
11998	2.286	0.92	0.77
274	1.749	1.067	0.984
11407	4.119	0.941	1.086
46	**0.656**	0.689	0.916
11326	1.294	0.585	1.089
11335	0.904	**0.284**	0.611
12197	1.769	0.595	0.938
11693	1.41	0.297	0.954
11780	2.098	1.251	2.029
12502	1.618	0.663	1.003
11866	0.802	0.827	0.748
11400	2.756	0.542	0.836
12256	1.971	0.383	0.875
10320	1.713	0.655	0.656
10030	1.38	0.448	0.54

8.8.3 Dataset 3 - Experiment results

Table 8.25: Metrics from experiment, dataset-3, sarima

Category ID	MASE	sMAPE	MASE-7
12322	3.159	0.33	1.507
11428	0.766	0.164	**0.568**
11850	2.632	0.53	0.893
11852	3.414	0.349	0.917
273	2.337	0.536	1.63
11036	**0.614**	**0.148**	0.688
11213	1.362	0.485	0.962
12532	1.217	0.512	0.681

Table 8.26: Metrics from experiment, dataset-3, local univariate lstm

Category ID	MASE	sMAPE	MASE-7
12322	2.186	**0.242**	1.026
11428	1.906	0.343	1.347
11850	1.56	0.368	**0.556**
11852	2.843	0.311	0.78
273	2.371	0.561	1.595
11036	3.001	0.539	3.223
11213	**1.318**	0.469	0.889
12532	1.898	0.985	1.032

Table 8.27: Metrics from experiment, dataset-3, global univariate lstm

Category ID	MASE	sMAPE	MASE-7
12322	2.035	**0.227**	0.961
11428	1.361	0.261	0.99
11850	2.779	0.558	0.924
11852	3.67	0.381	0.994
273	**1.315**	0.366	**0.875**
11036	1.432	0.306	1.545
11213	1.549	0.544	1.123
12532	1.607	0.76	0.894

Table 8.28: Metrics from experiment, dataset-3, local multivariate lstm

Category ID	MASE	sMAPE	MASE-7
12322	**0.807**	**0.1**	0.381
11428	2.147	0.57	1.606
11850	1.017	0.248	**0.345**
11852	1.393	0.171	0.363
273	2.127	0.522	1.447
11036	2.125	0.412	2.343
11213	1.096	0.393	0.757
12532	1.249	0.538	0.665

Table 8.29: Metrics from experiment, dataset-3, global multivariate lstm

Category ID	MASE	sMAPE	MASE-7
12322	1.679	0.192	0.813
11428	**0.814**	**0.17**	**0.639**
11850	2.683	0.544	0.882
11852	3.306	0.351	0.898
273	1.341	0.372	0.903
11036	0.989	0.223	1.138
11213	1.175	0.405	0.91
12532	1.839	0.936	1.007

Table 8.30: Metrics from experiment, dataset-3, local univariate cnn ae lstm

Category ID	MASE	sMAPE	MASE-7
12322	2.165	**0.24**	1.019
11428	2.181	0.382	1.554
11850	**1.565**	0.369	**0.561**
11852	2.845	0.311	0.785
273	2.212	0.534	1.486
11036	2.113	0.412	2.177
11213	1.838	0.604	1.285
12532	1.93	1.013	1.068

Table 8.31: Metrics from experiment, dataset-3, global univariate cnn ae lstm

Category ID	MASE	sMAPE	MASE-7
12322	1.997	**0.223**	0.943
11428	**1.155**	0.228	0.836
11850	2.733	0.551	0.902
11852	3.863	0.397	1.041
273	1.244	0.351	**0.821**
11036	1.392	0.298	1.496
11213	1.564	0.549	1.115
12532	1.575	0.738	0.875

Table 8.32: Metrics from experiment, dataset-3, local multivariate cnn ae lstm

Category ID	MASE	sMAPE	MASE-7
12322	2.06	0.284	0.927
11428	0.791	0.169	0.583
11850	1.178	0.296	0.38
11852	1.23	**0.152**	**0.323**
273	1.621	0.421	1.1
11036	**0.749**	0.17	0.836
11213	1.532	0.536	1.078
12532	1.658	0.808	0.891

Table 8.33: Metrics from experiment, dataset-3, global multivariate cnn ae lstm

Category ID	MASE	sMAPE	MASE-7
12322	1.784	**0.201**	**0.886**
11428	**1.503**	0.283	1.044
11850	3.502	0.635	1.388
11852	5.4	0.501	1.421
273	3.002	0.657	2.049
11036	4.733	0.737	5.208
11213	1.739	0.59	1.17
12532	1.82	0.927	1.004

8.8.4 Dataset with variance - Additional Experiments

Table 8.34: Metrics from experiment, dataset-variance, dataset-high-variance-cnn-ae-lstm-local-univariate

Category ID	MASE	sMAPE	MASE-7
77	0.516	1.242	0.711
79	0.88	**0.616**	1.394
11217	**0.506**	0.905	**0.499**
11334	0.727	0.691	0.728
11992	0.524	0.837	0.714
12265	0.553	0.888	0.785
12511	0.729	0.645	0.645

Table 8.35: Metrics from experiment, dataset-variance, dataset-high-variance-lstm-local-univariate

Category ID	MASE	sMAPE	MASE-7
77	0.519	1.349	0.705
79	0.876	**0.613**	1.394
11217	**0.517**	0.897	**0.536**
11334	0.798	0.788	0.811
11992	0.526	0.839	0.718
12265	0.524	0.879	0.732
12511	0.752	0.681	0.676

Table 8.36: Metrics from experiment, dataset-variance, low-variance-cnn-ae-lstm-local-univariate

Category ID	MASE	sMAPE	MASE-7
10053	5.263	0.626	1.133
11037	6.043	0.616	1.132
11041	9.286	0.935	1.048
11048	**1.499**	**0.315**	**0.689**
11456	6.149	0.675	1.104
11817	2.053	0.595	0.936
13323	5.213	0.723	1.531

Table 8.37: Metrics from experiment, dataset-variance, low-variance-cnn-ae-lstm-local-univariate

Category ID	MASE	sMAPE	MASE-7
10053	5.263	0.626	1.133
11037	6.043	0.616	1.132
11041	9.286	0.935	1.048
11048	**1.499**	**0.315**	**0.689**
11456	6.149	0.675	1.104
11817	2.053	0.595	0.936
13323	5.213	0.723	1.531

Table 8.38: Metrics from experiment, dataset-variance, ok-variance-lstm-local-univariate

Category ID	MASE	sMAPE	MASE-7
16	0.731	0.27	**0.568**
17	1.306	1.2	0.666
28	0.628	0.242	0.689
34	0.697	**0.239**	0.596
40	0.919	0.414	0.878
44	**0.568**	0.245	0.685
45	0.706	0.611	1.005
48	0.75	0.799	0.971

Table 8.39: Metrics from experiment, dataset-variance, ok-variance-lstm-local-univariate

Category ID	MASE	sMAPE	MASE-7
16	0.731	0.27	**0.568**
17	1.306	1.2	0.666
28	0.628	0.242	0.689
34	0.697	**0.239**	0.596
40	0.919	0.414	0.878
44	**0.568**	0.245	0.685
45	0.706	0.611	1.005
48	0.75	0.799	0.971

8.8.5 Dataset with differencing - Additional Experiments

Table 8.40: Metrics from experiment, dataset-diff, local univariate lstm dataset 1

Category ID	MASE	sMAPE	MASE-7
2	1.144	0.151	1.805
6	0.748	0.244	1.346
9	0.597	0.097	0.529
10	0.659	0.161	1.21
11	0.901	0.225	0.64
13	**0.553**	0.081	0.985
20	0.637	0.346	0.576
22	0.845	0.178	1.046
24	1.743	0.133	1.496
26	1.049	0.186	**0.407**
27	1.661	0.178	0.599
28	1.5	0.466	1.489
29	2.124	0.192	1.211
32	1.154	0.204	0.593
33	1.704	0.238	1.397
34	0.725	0.25	0.637
39	1.314	0.393	0.92
41	0.674	0.286	1.436
51	0.568	**0.064**	0.95
54	2.277	0.088	1.062

Table 8.41: Metrics from experiment, dataset-diff, local univariate lstm dataset 1 diff

Category ID	MASE	sMAPE	MASE-7
2	1.206	0.173	2.072
6	0.697	0.222	1.296
9	0.929	0.159	0.662
10	0.645	0.158	1.176
11	0.829	0.212	0.49
13	0.765	0.114	0.863
20	0.684	0.369	0.688
22	0.717	0.155	0.835
24	2.441	0.202	1.735
26	1.176	0.216	**0.37**
27	3.015	0.355	1.022
28	1.105	0.371	0.69
29	3.199	0.321	1.177
32	0.894	0.163	0.48
33	0.583	0.09	0.661
34	0.751	0.263	0.602
39	1.742	0.573	1.178
41	**0.58**	0.243	1.246
51	2.69	0.357	1.497
54	2.166	**0.084**	0.861

Table 8.42: Metrics from experiment, dataset-diff, local univariate lstm dataset 3

Category ID	MASE	sMAPE	MASE-7
12322	2.148	**0.238**	1.012
11428	3.041	0.492	2.213
11850	1.642	0.383	**0.583**
11852	2.717	0.3	0.761
273	2.434	0.571	1.628
11036	2.515	0.477	2.636
11213	**1.234**	0.437	0.878
12532	1.923	1.009	1.046

Table 8.43: Metrics from experiment, dataset-diff, local univariate lstm dataset 3 diff

Category ID	MASE	sMAPE	MASE-7
12322	1.86	**0.209**	1.083
11428	1.302	0.252	0.858
11850	3.105	0.597	1.066
11852	4.401	0.436	1.197
273	**0.683**	0.217	**0.551**
11036	0.972	0.22	1.082
11213	1.36	0.484	0.804
12532	1.243	0.525	0.684

8.8.6 T-test statistical significance test results

Table 8.44: Student t-test, measuring confidence of significant difference between predictions, statistic value. MASE error - p-value

	local-univariate	local-univariate
Differencing dataset	0.18392	0.50322

Table 8.45: Student t-test, measuring confidence of significant difference between LSTM models all dataset, statistic value. MASE error. l-u = local univariate, l-m= local multivariate, g-u = global univariate etc. - p-value

	l-u	l-m	g-u	g-m
l-u	nan	**0.00321**	0.6342	0.12963
l-m	**0.00321**	nan	**0.04453**	0.18605
g-u	0.6342	**0.04453**	nan	**0.00858**
g-m	0.12963	0.18605	**0.00858**	nan

Table 8.46: Student t-test, measuring confidence of significant difference between the local Univariate LSTM and other LSTM models on dataset 1. MASE error - p-value

	l-u	l-m	g-u	g-m
l-u	nan	**0.04385**	0.83216	0.7314
l-m	**0.04385**	nan	0.07791	0.14276
g-u	0.83216	0.07791	nan	0.66554
g-m	0.7314	0.14276	0.66554	nan

Table 8.47: Student t-test, measuring confidence of significant difference between LSTM models dataset 2, statistic value. MASE error - p-value

	l-u	l-m	g-u	g-m
l-u	nan	0.10908	0.82966	0.31324
l-m	0.10908	nan	0.22664	0.4502
g-u	0.82966	0.22664	nan	0.06325
g-m	0.31324	0.4502	0.06325	nan

Table 8.48: Student t-test, measuring confidence of significant difference between LSTM models dataset 3, statistic value. MASE error - p-value

	l-u	l-m	g-u	g-m
l-u	nan	**0.01711**	0.62535	0.2787
l-m	**0.01711**	nan	0.3017	0.61195
g-u	0.62535	0.3017	nan	**0.03802**
g-m	0.2787	0.61195	**0.03802**	nan

Table 8.49: Student t-test, measuring confidence of significant difference between the local Univariate LSTM and other LSTM models on dataset 1. sMape error - p-value

	l-u	l-m	g-u	g-m
l-u	nan	**0.03921**	0.71281	0.64284
l-m	**0.03921**	nan	0.05036	0.07294
g-u	0.71281	0.05036	nan	0.85439
g-m	0.64284	0.07294	0.85439	nan

Table 8.50: Student t-test, measuring confidence of significant difference between LSTM models dataset 2, statistic value. sMape error - p-value

	l-u	l-m	g-u	g-m
l-u	nan	0.22887	0.63831	0.32022
l-m	0.22887	nan	0.29329	0.40499
g-u	0.63831	0.29329	nan	0.31726
g-m	0.32022	0.40499	0.31726	nan

Table 8.51: Student t-test, measuring confidence of significant difference between LSTM models dataset 3, statistic value. sMape error - p-value

	l-u	l-m	g-u	g-m
l-u	nan	0.13986	0.38429	0.18512
l-m	0.13986	nan	0.49114	0.76066
g-u	0.38429	0.49114	nan	0.45713
g-m	0.18512	0.76066	0.45713	nan

Table 8.52: Student t-test, measuring confidence of significant difference between predictions, statistic value. MASE error - p-value

	local_multivariate	global_univariate	global_multivariate
dataset 1	0.043850	0.832161	0.731403
dataset 2	0.109082	0.829658	0.313239
dataset 3	0.017108	0.625346	0.278702

Table 8.53: Student t-test, measuring confidence of significant difference between predictions, comparing LSTM model structures against local univariate LSTM. sMape error - p-value

	local_multivariate	global_univariate	global_multivariate
dataset 1	0.039211	0.712811	0.642842
dataset 2	0.228875	0.638309	0.320224
dataset 3	0.139856	0.384286	0.185118

Table 8.54: Student t-test, measuring confidence of significant difference between LSTM models all dataset, statistic value. sMape error. l-u = local univariate, l-m= local multivariate, g-u = global univariate etc. - p-value

	l-u	l-m	g-u	g-m
l-u	nan	**0.03918**	0.36293	0.10352
l-m	**0.03918**	nan	0.10993	0.23882
g-u	0.36293	0.10993	nan	0.19305
g-m	0.10352	0.23882	0.19305	nan

Table 8.55: Student t-test, measuring confidence of significant difference between predictions on the CNN-AE-LSTM and the LSTM for different model structures. MASE error - p-value

	local-univariate	local-multivariate	global-univariate	global-multivariate
dataset 1	0.93845	0.51339	**0.00053**	0.78276
dataset 2	0.48063	**0.0499**	0.1826	**0.02548**
dataset 3	0.84437	0.67088	0.49328	**0.0295**

Figure 8.2: Test predictions for local univariate LSTM on category 11850 with last week values ploted as a reference.

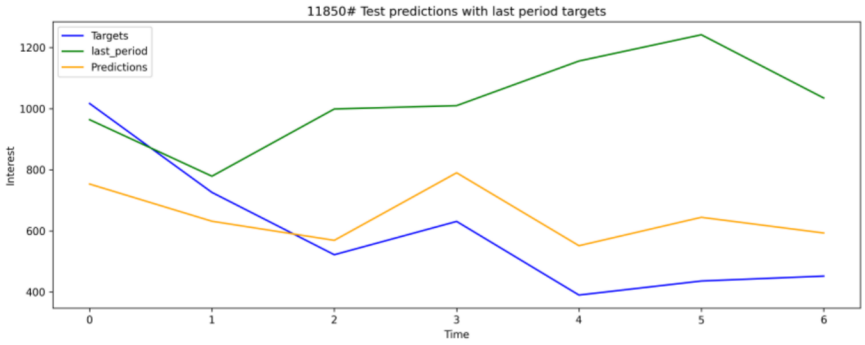
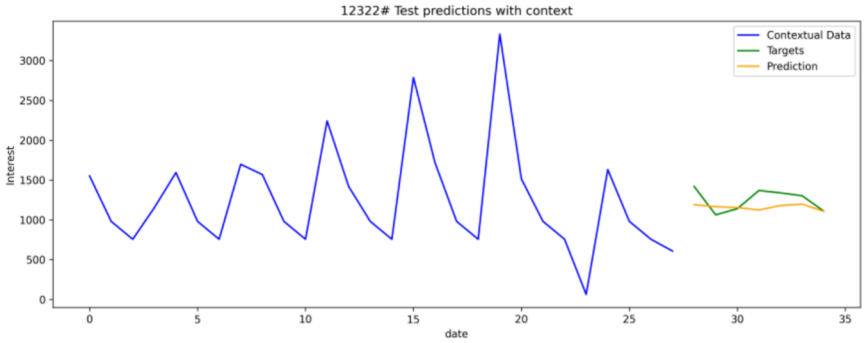
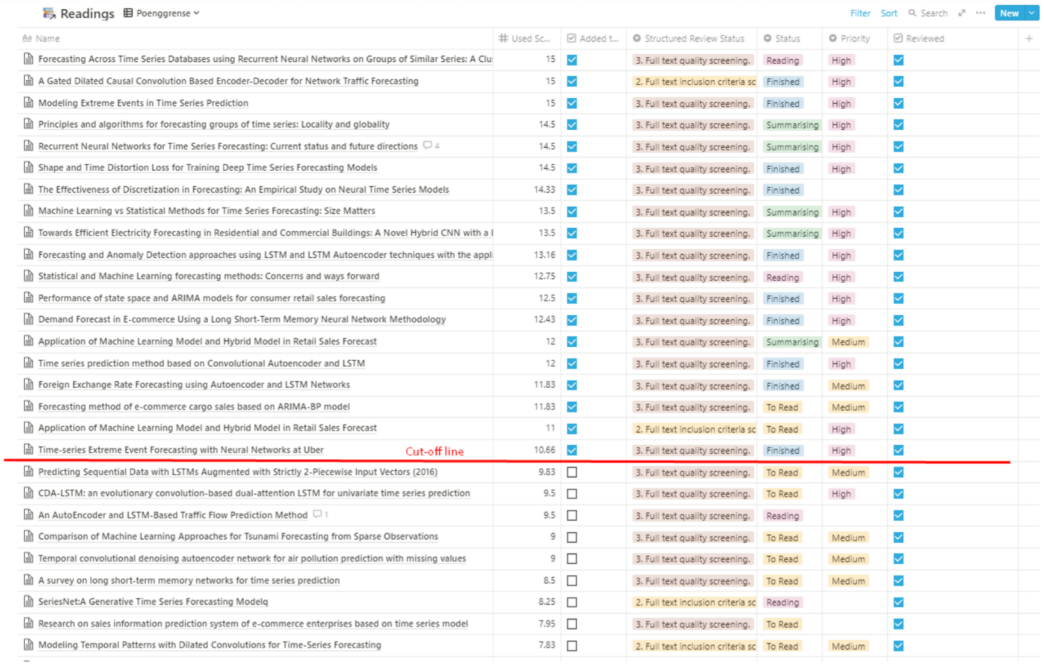


Figure 8.3: Test predictions for local univariate LSTM on category 12322 with last week values ploted as a context.



8.9 Structured Literature Review cut-off line



Name	Used Sc...	Added t...	Structured Review Status	Status	Priority	Reviewed
Forecasting Across Time Series Databases using Recurrent Neural Networks on Groups of Similar Series: A Clu	15	☑	3. Full text quality screening.	Reading	High	☑
A Gated Dilated Causal Convolution Based Encoder-Decoder for Network Traffic Forecasting	15	☑	2. Full text inclusion criteria sc	Finished	High	☑
Modeling Extreme Events in Time Series Prediction	15	☑	3. Full text quality screening.	Finished	High	☑
Principles and algorithms for forecasting groups of time series: Locality and globality	14.5	☑	3. Full text quality screening.	Summarising	High	☑
Recurrent Neural Networks for Time Series Forecasting: Current status and future directions 4	14.5	☑	3. Full text quality screening.	Summarising	High	☑
Shape and Time Distortion Loss for Training Deep Time Series Forecasting Models	14.5	☑	3. Full text quality screening.	Finished	High	☑
The Effectiveness of Discretization in Forecasting: An Empirical Study on Neural Time Series Models	14.33	☑	3. Full text quality screening.	Finished		☑
Machine Learning vs Statistical Methods for Time Series Forecasting: Size Matters	13.5	☑	3. Full text quality screening.	Summarising	High	☑
Towards Efficient Electricity Forecasting in Residential and Commercial Buildings: A Novel Hybrid CNN with a l	13.5	☑	3. Full text quality screening.	Summarising	High	☑
Forecasting and Anomaly Detection approaches using LSTM and LSTM Autoencoder techniques with the appl	13.16	☑	3. Full text quality screening.	Finished	High	☑
Statistical and Machine Learning forecasting methods: Concerns and ways forward	12.75	☑	3. Full text quality screening.	Reading	High	☑
Performance of state space and ARIMA models for consumer retail sales forecasting	12.5	☑	3. Full text quality screening.	Finished	High	☑
Demand Forecast in E-commerce Using a Long Short-Term Memory Neural Network Methodology	12.43	☑	3. Full text quality screening.	Finished	High	☑
Application of Machine Learning Model and Hybrid Model in Retail Sales Forecast	12	☑	3. Full text quality screening.	Summarising	Medium	☑
Time series prediction method based on Convolutional Autoencoder and LSTM	12	☑	3. Full text quality screening.	Finished	High	☑
Foreign Exchange Rate Forecasting using Autoencoder and LSTM Networks	11.83	☑	3. Full text quality screening.	Finished	Medium	☑
Forecasting method of e-commerce cargo sales based on ARIMA-BP model	11.83	☑	3. Full text quality screening.	To Read	Medium	☑
Application of Machine Learning Model and Hybrid Model in Retail Sales Forecast	11	☑	2. Full text inclusion criteria sc	To Read	High	☑
Time-series Extreme Event Forecasting with Neural Networks at Uber	10.66	☑	3. Full text quality screening.	Finished	High	☑
Predicting Sequential Data with LSTMs Augmented with Strictly 2-Piecewise Input Vectors (2016)	9.83	☐	3. Full text quality screening.	To Read	Medium	☑
CDA-LSTM: an evolutionary convolution-based dual-attention LSTM for univariate time series prediction	9.5	☐	3. Full text quality screening.	To Read	High	☑
An AutoEncoder and LSTM-Based Traffic Flow Prediction Method 1	9.5	☐	3. Full text quality screening.	Reading		☑
Comparison of Machine Learning Approaches for Tsunami Forecasting from Sparse Observations	9	☐	3. Full text quality screening.	To Read	Medium	☑
Temporal convolutional denoising autoencoder network for air pollution prediction with missing values	9	☐	3. Full text quality screening.	To Read	Medium	☑
A survey on long short-term memory networks for time series prediction	8.5	☐	3. Full text quality screening.	To Read	Medium	☑
SeriesNet:A Generative Time Series Forecasting Modelq	8.25	☐	2. Full text inclusion criteria sc	Reading		☑
Research on sales information prediction system of e-commerce enterprises based on time series model	7.95	☐	3. Full text quality screening.	To Read		☑
Modeling Temporal Patterns with Dilated Convolutions for Time-Series Forecasting	7.83	☐	2. Full text inclusion criteria sc	To Read	Medium	☑

Figure 8.4: The line we drew to choose which papers to include

Sindre J. I. Sivertsen, Sander K. Kilen

Exploring a Convolutional Autoencoder with LSTM on E-commerce Time-Series Forecasting

NTNU
Norwegian University of
Science and Technology