TIME SERIES FORECASTING USING MACHINE LEARNING FOR NETWORK COMMUNICATION

VYUŽITÍ STROJOVÉHO UČENÍ PRO PREDIKCI ČASOVÝCH ŘAD U POČÍTAČOVÉ KOMUNIKACE

MASTER'S THESIS
DIPLOMOVÁ PRÁCE

AUTHOR **Bc. ALEŠ KAŠPÁREK**
AUTOR PRÁCE

SUPERVISOR **doc. Ing. PETR MATOUŠEK, Ph.D., M.A.**
VEDOUCÍ PRÁCE

BRNO 2023

BRNO UNIVERSITY OF TECHNOLOGY FACULTY OF INFORMATION TECHNOLOGY

Master's Thesis Assignment

153380

Institut:	Department of Information Systems (DIFS)
Student:	**Kašpárek Aleš, Bc.**
Programme:	Information Technology and Artificial Intelligence
Specialization:	Machine Learning
Title:	**Time Series Forecasting Using Maching Learning for Network Communication**
Category:	Networking
Academic year:	2023/24

Assignment:

1. Describe the state of the art of machine learning models used for time series forecasting. Discuss time series forecasting challenges.
2. Based on recommendation of your supervisor, prepare datasets with network communication with various time series behavior.
3. Apply selected machine learning models on various time series datasets to predict the future.
4. Analyze the results and find the optimum models. Compare your results with other published approaches.
5. Discuss contribution of your work and application of the proposed approach to real-world communication.

Literature:

- Zhang Ying, Bernard J. Jansen, and Amanda Spink. "Time series analysis of a Web search engine transaction log." *Information processing & management* 45.2 (2009): 230-245.
- Shai Shalev-Shwartz, Shai Ben-David, Understanding Machine Learning: From Theory to Algorithms. ISBN:1107057132, Pages: 397, Year: 2014
- Aileen Nielsen, Practical Time Series Analysis : Prediction with Statistics and Machine Learning. ISBN:1492041653, Pages: 400, 2019.
- Francesca Lazzeri: Machine Learning for Time Series Forecasting with Python, Wiley, 2021.
- Mohammad Braei and Sebastian Wagner: Anomaly Detection in Univariate Time-series: A Survey on the State-of-the-Art, arXiv, 2004.00433, 2020.
- Vitor Cerqueira and Luis Torgo and Carlos Soares: Machine Learning vs Statistical Methods for Time Series Forecasting: Size Matters, arXiv, 1909.13316, 2019.
- Samuel Budai: Analýza časových řad, BP, FIT VUT v Brně, 2023.

Requirements for the semestral defence:
Points 1-3.

Detailed formal requirements can be found at https://www.fit.vut.cz/study/theses/

Supervisor:	**Matoušek Petr, doc. Ing., Ph.D., M.A.**
Consultant:	Dr. Nabhan Khatib, AT&T
Head of Department:	Kolář Dušan, doc. Dr. Ing.
Beginning of work:	1.11.2023
Submission deadline:	17.5.2024
Approval date:	30.10.2023

Abstract

This master thesis examines the complex world of network communication systems, which require advanced forecasting methods to run efficiently, reliably, and safely. With networks becoming more complex, accurately predicting network conditions and traffic is critical for planning, resource management, detecting unusual activity, and improving systems.

The thesis commences by introducing the concept of time series data, laying the foundation for understanding the temporal dynamics within network systems. It progresses by presenting an array of analytical tools and techniques for dissecting this kind of data, with a particular focus on traditional statistical methods. Among these, the Moving Average (MA), Auto Regressive (AR) and Auto Regresive Integrated Moving Average (ARIMA) models are given special attention for its established capabilities in forecasting.

The shift from traditional forecasting to the use of machine learning (ML) is central to this thesis. It investigates several machine learning (ML) approaches, such as Long Short-Term Memory (LSTM) networks, convolutional neural networks (CNNs), to demonstrate how they can identify the complex patterns in network traffic.

Abstrakt

Tato diplomová práce zkoumá komplexní svět síťových komunikačních systémů, které vyžadují pokročilé metody předpovědi, aby fungovaly efektivně, spolehlivě a bezpečně. Se sítěmi stále složitější, přesné předvídání podmínek sítě a jejího provozu je rozhodující pro plánování, řízení zdrojů, detekci anomálií a zlepšování systémů.

Práce začíná představením konceptu časových řad dat, který pokládá základ pro pochopení dynamiky v síťových systémech. Pokračuje tím, že představuje řadu analytických nástrojů a technik pro rozbor tohoto druhu dat, se zvláštním zaměřením na tradiční statistické metody. Mezi nimi je modelům Moving Average (MA), Auto Regressive (AR) a Auto Regresive Integrated Moving Average (ARIMA) věnována zvláštní pozornost pro své schopnosti v předpovídání budoucích stavů.

Posun od tradičního předpovídání k používání strojového učení (ML) je ústředním bodem této práce. Práce zkoumá několik přístupů strojového učení (ML), jako jsou sítě Long Short-Term Memory (LSTM), konvoluční neuronové sítě (CNN), aby ukázala, jak mohou tyto metody identifikovat složité vzorce v síťovém provozu.

Keywords

time series, machine learning, forecast, convolutional neural networks, Long Short-Term Memory, ARIMA

Klíčová slova

časové řady, strojové učení, předpovědi, koncoluční neuronové sítě, Long Short-Term Memory, ARIMA

Reference

KAŠPÁREK, Aleš. *Time Series Forecasting Using Machine Learning for Network Communication.* Brno, 2023. Master's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor doc. Ing. Petr Matoušek, Ph.D., M.A.

Rozšířený abstrakt

Tato diplomová práce se zaměřuje na aplikaci strojového učení pro předpovědi časových řad v oblasti síťové komunikace. V dnešní době, kdy síťové systémy nabývají na komplexnosti a dynamice, je klíčové předvídat síťové podmínky a provoz s vysokou mírou přesnosti. Přesné předpovědi jsou zásadní pro efektivní plánování, řízení zdrojů, detekci neobvyklé aktivity a obecné zlepšení systémů. Práce zdůrazňuje význam a nutnost rozvoje a implementace pokročilých prediktivních modelů, které mohou zvládnout vysoké požadavky moderních komunikačních sítí.

Základem práce je komplexní přehled a aplikace časových řad a přístupů pro jejich analýzu. Práce začíná revizí základních statistických metod předpovědi, jako jsou modely klouzavých průměrů (MA), autoregresní modely (AR) a modely autoregresního integrovaného klouzavého průměru (ARIMA). Důležitou částí je přechod k moderním technikám strojového učení, kde jsou využity sítě Long Short-Term Memory (LSTM) a konvoluční neuronové sítě (CNN), které jsou implementovány a porovnány vůči tradičním metodám.

V praktické části je provedena implementace vybraných modelů na reálné datové sadě síťového provozu. Výsledky ukázaly, že modely strojového učení poskytují významné zlepšení v přesnosti předpovědí díky jejich schopnosti identifikovat a učit se ze složitých vzorců v datech, což tradiční metody nedokáží. LSTM sítě byly zvláště účinné v zachycení dlouhodobých závislostí v časových řadách, zatímco CNN modely excelovaly v rychlé a efektivní identifikaci vzorců.

Analýza a výsledky práce mají dalekosáhlé důsledky pro řízení a optimalizaci síťových systémů. Efektivní využití prediktivních modelů umožňuje lepší přizpůsobení síťových kapacit aktuálním potřebám, což vede k optimalizaci zdrojů a zlepšení kvality služeb. Předpovědi pomáhají předcházet výpadkům a identifikovat bezpečnostní hrozby dříve, než mohou způsobit škodu.

Práce předkládá důkazy, že strojové učení může radikálně transformovat metody předpovídání v síťové komunikaci, poskytuje metodologický rámec pro jejich implementaci a ukazuje cestu pro další výzkum v této dynamicky se rozvíjející oblasti. Díky těmto pokrokům mohou síťoví operátoři lépe reagovat na výzvy spojené s růstem a evolucí digitálních komunikačních technologií.

Time Series Forecasting Using Machine Learning for Network Communication

Declaration

I hereby declare that this Bachelor's thesis was prepared as an original work by the author under the supervision of doc. Ing. Petr Matoušek, Ph.D., M.A. The supplementary information was provided by Ing. Nabhan Khatib, Ph.D. I have listed all the literary sources, publications and other sources, which were used during the preparation of this thesis.

. . . .
Aleš Kašpárek
May 10, 2024

Acknowledgements

I would like to express my gratitude to my supervisor doc. Ing. Petr Matoušek, Ph.D., M.A., for providing me with information necessary to accomplish this project and guiding me during the creation of this thesis.

Contents

List of Figures

Chapter 1

Introduction

The rapid expansion of digital communication networks has highlighted the vital need for better predictive models to assure the networks durability, efficiency, and scalability. Time series forecasting is critical in network communication because it allows for the prediction of network loads, bandwidth requirements, and possible bottlenecks, resulting in proactive network management and optimization. Traditional statistical approaches have long been the backbone of forecasting because of their sound theoretical underpinnings and interpretability. However, the introduction of machine learning (ML) models, which are noted for their flexibility and capacity to manage massive amounts of data, has transformed forecasting approaches, creating new difficulties and possibilities. The purpose of this thesis is to undertake a comprehensive comparison of statistical and machine learning approaches for time series forecasting in network communications.

The importance of forecasting in network communications cannot be emphasized. It is essential for traffic control, quality of service (QoS) provisioning, network planning, and security. Accurate projections enable network operators to make educated decisions, maximizing resource allocation and addressing possible issues before they affect customers. Given the dynamic and frequently non-linear nature of network traffic, the investigation of various forecasting approaches becomes critical.

Statistical approaches, such as ARIMA (AutoRegressive Integrated Moving Average) and Exponential Smoothing, have long been considered the gold standard for time series forecasting due to their efficiency and interpretative insights into data. These approaches, which are based on statistical theory, excel in capturing linear correlations and seasonal patterns in time series data. However, they frequently fall short when dealing with non-linear patterns, complicated relationships, or data with no obvious temporal structure.

Machine learning approaches, such as neural networks, provide a viable alternative that can represent complicated non-linear interactions without the need for explicit model formulation. These approaches have yielded encouraging results in a variety of forecasting areas, including network communications, by exploiting their ability to learn from big datasets and catch complex patterns. Nonetheless, ML approaches provide their own set of issues, such as the requirement for large amounts of data for training, the potential of overfitting, and, in many cases, a lack of transparency in the decision-making process.

This thesis will explore these two classes of forecasting methods within the context of network communications. It will assess their predictive accuracy for different forecasting scenarios encountered in network traffic.

In pursuit of this objective, the thesis will first review the fundamental principles underlying statistical and machine learning forecasting methods. It will then delve into the specific challenges of time series forecasting in network communications, followed by an empirical comparison of selected statistical and ML models on benchmark datasets.

Chapter 2

Preliminaries

This chapter provides a foundational overview of time series data and time series analysis. By dissecting the nature and characteristics of time series, it aims to establish a solid foundation for applying advanced statistical and machine learning techniques in later chapters. The techniques covered include exploratory data analysis, correlation analysis and tests for stationarity, which will be applied in Chapter 4. These methods lay the groundwork for understanding the behavior and characteristics of time series data, which is critical for effective modeling and forecasting.

2.1 Time series

According to [3], a time series is a set of observations x_t , each one being recorded at a specific time t. A discrete-time time series is one in which the set T_0 of times at which observations are made is a discrete set, as is the case, for example, when observations are made at fixed time intervals. Continuous time series is obtained when observations are recorded continuously over some time interval, e.g., when $T_0 = [0, 1]$.

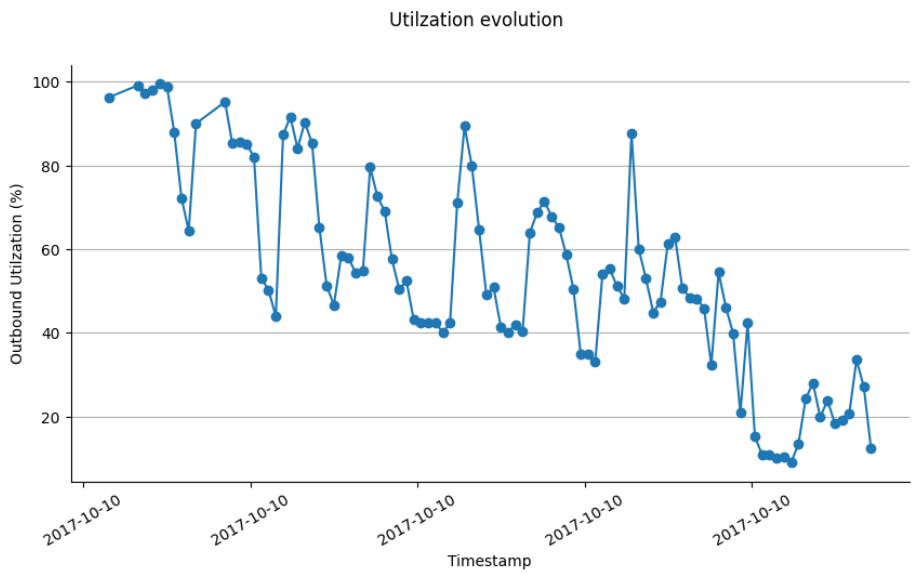

Figure 2.1: Example of time series.

2.2 Time series analysis

Time series analysis is a collection of statistical techniques for analyzing time-ordered data points. Understanding the underlying patterns and structures in time series data will allow us to make predictions about future values. This section will look at several methods for analyzing time series, each with its own set of applications, assumptions, and constraints.

Following methods are crucial in time series analysis for identifying patterns, trends and anomalies, which guide the model selection and improves forecasting accuracy. Time series analysis is instrumental in improving understanding and utilization of sequential data. It aids in spotting data quality issues like outliers and missing values, clarifying the distribution of data over time, and revealing important characteristics such as trends and seasonality. Furthermore, it is crucial for assessing whether data remains consistent over time, known as stationarity, which is a key consideration in many statistical modeling and forecasting techniques. Essentially, it helps ensure the reliability of data-driven decisions by providing a clearer picture of the underlying patterns and stability of the data.

2.2.1 Exploratory Analysis

Exploratory Analysis of time series data is an initial approach to analyzing temporal datasets. It involves a series of methods aimed at understanding the underlying structure and components of the time series, summarizing its main characteristics, and uncovering any inherent patterns such as trends, seasonality, and anomalies.

Plotting Time Series Data

In data analysis, time series plotting is an essential tool that performs a number of vital tasks. It offers a clear visual representation that frequently reveals patterns, trends, and anomalies that are not visible from raw data alone, and it is the first step towards understanding the underlying structure of the data. A visual assessment can quickly reveal important components that are essential for any further research or forecasting, such as patterns, seasonality, and cyclical variations. Plotting also aids in the identification of outliers and sudden changes in the behavior of the data, which may indicate significant occurrences or modifications in the system under observation. Without plots, drawing conclusions from time series data would be far more difficult and counter intuitive, which would seriously impede subsequent analysis and decision-making [12].

Line Plots

This is the most common type of plot used for time series data. Information is shown as a set of data points known as „markers“ that are joined by segments of straight lines. It is very helpful for displaying trends over time.

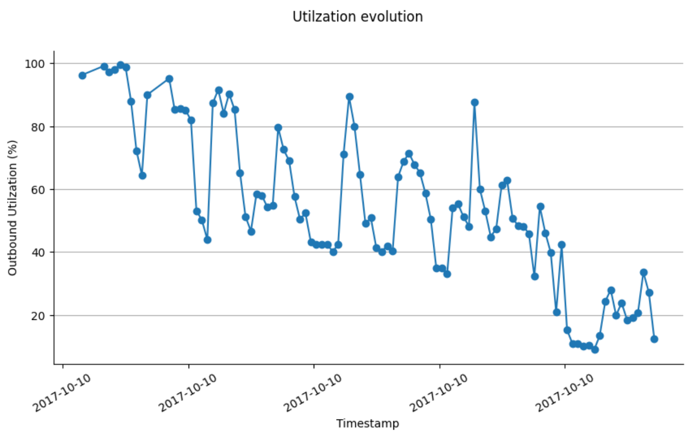

Figure 2.2: Example of line plot.

Scatter Plots

Scatter plots can be helpful to highlight individual data points, even though they are less common for time series because they do not emphasize the sequential nature of the data. This is especially true if we are interested in the distribution or density of the points over time.

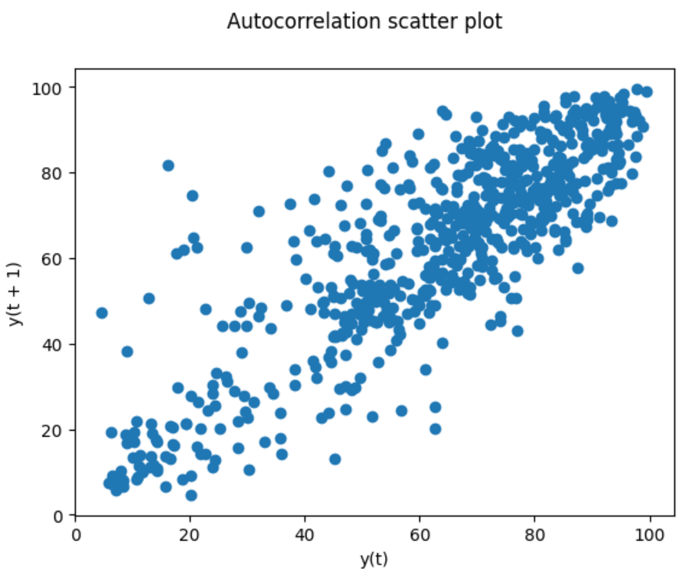

Figure 2.3: Example of scatter plot.

Autocorrelation and Partial Autocorrelation Plots

These are specific time series analysis plots that display the series correlation with itself at various lags. They play a vital role in determining the autoregressive (AR) processes order.

2.2.2 Moving Averages

This method provides the foundation for smoothing time series data, efficiently decreasing noise and revealing underlying trends. Moving averages are produced by taking the average of a certain number of consecutive data points over a specified time period, and as time passes, the average „moves“ by discarding the oldest data point and inserting a new one. This method can take several forms, including the simple moving average (SMA), which gives equal weight to all data points, and the exponential moving average (EMA), which gives more weight to recent data [11].

2.2.3 Decomposition

This technique divides a time series into three distinct components: trend, seasonality, and residuals. The trend component depicts the data long-term evolution, exhibiting directional moves that are not constrained by a temporal range. Seasonality refers to regular, predictable patterns that repeat across time, such as daily, monthly, or annual cycles. Finally, the residuals, often known as the random component, include the anomalies and noise that are not explained by the trend or seasonality [12].

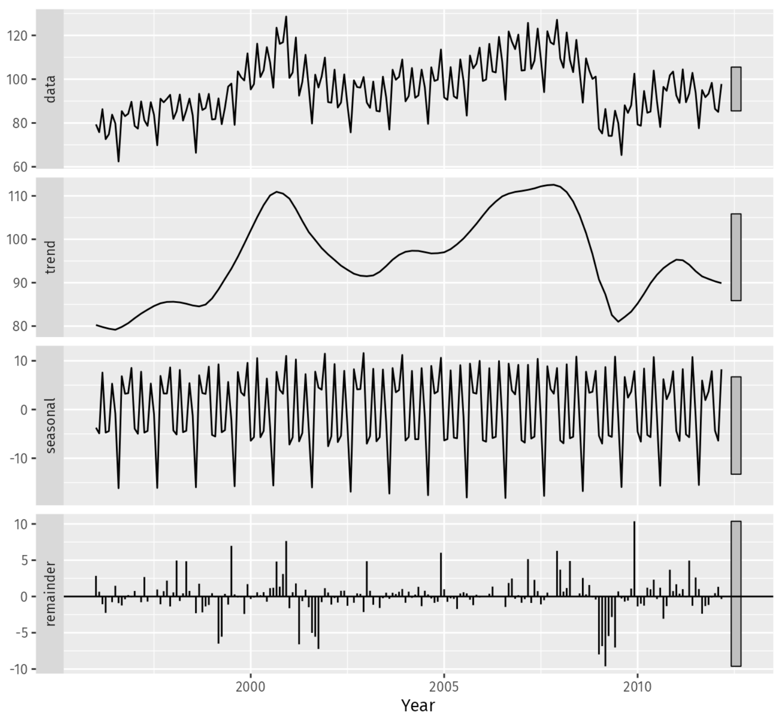

Figure 2.4: Example of time series decomposition [12].

2.2.4 Stationarity Testing

A simple definition of a stationary process is the following: a process is stationary if for all possible lags (lag refers to the position shift used to compare the series with its past values), k, the distribution of $y_t, y_{t+1}, ..., y_{t+k}$, does not depend on t[19].

Time series data frequently requires that the data is stationary, which means that its statistical properties do not change over time. These methods include:

- Unit Root Tests - the Augmented Dickey-Fuller (ADF) test is used to check for stationarity in time series by determining the presence of a unit root. This method tests the null hypothesis H_0 that there exists a unit root in the time series and it is non-stationary [19].
- KPSS Test - the Kwiatkowski-Phillips-Schmidt-Shin (KPSS) test looks for stationarity in the vicinity of a deterministic trend [19].

While the ADF test posits a null hypothesis of a unit root, the Kwiatkowski-Phillips-Schmidt-Shin (KPSS) test posits a null hypothesis of a stationary process [19].

2.2.5 Correlation Analysis

Correlation analysis aids in determining the relationship between time series observations taken at different points in time. Among these methods are:

- Autocorrelation Function (ACF) is an important tool in time series analysis, which calculates the correlation between observations made at various time lags and effectively illustrates the relationship between a data point and its historical values. In time series data, it aids in spotting recurring patterns or serial dependencies. It is especially helpful in determining whether historical values have a linear predictive relationship with future values. The ACF plot, which shows the presence of seasonality or cyclical patterns, graphically depicts the degree of correlation at various time intervals with lags on the horizontal axis and correlation coefficients on the vertical. Sharp cutoffs frequently indicate a more stationary series, whereas a slowly declining ACF shows a high level of autocorrelation, which is typical in data with a strong trend component. When choosing a model for time series forecasting, the analysis of ACF is essential since it guides the selection of parameters for ARIMA models and aids in the diagnosis of problems such as over-differencing or non-stationarity [18].
- Partial Autocorrelation Function (PACF) provide insights into the direct relationship between an observation and its lagged counterpart. In order to determine the order of autoregressive (AR) terms in ARIMA models, this function is essential in determining the degree of influence that a previous value, independent of other lagged values, has on a current value. A clearer view of temporal interdependence is provided by the PACF, which isolates the direct effects, in contrast to the ACF, which may show correlation due to indirect effects. Similar in format to the ACF plot, the PACF plot shows the partial correlation coefficients against various lags; notable spikes suggest possible AR terms for modeling. To ensure that the models in time series forecasting accurately capture the true nature of the underlying temporal dynamics, it is imperative to comprehend and interpret the PACF [18].

2.3 Time series performance metrics

In the field of statistical modeling and predictive analytics, accurately determining how well a model forecasts is crucial. Following methods are essential tools to quantitatively measure how far off a model predictions are from the actual observed values, providing a clear picture of accuracy and precision. Metrics are considered vital for the comparison of different machine learning models as they provide quantifiable measures of a model performance. For regression tasks, the difference between the predicted and actual values is quantified by metrics such as Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and Mean Absolute Error (MAE), which indicate the prediction accuracy of the model. Performance metrics such as Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and others are introduced to evaluate the accuracy and effectiveness of different models. These metrics will be used to evaluate and compare models in Chapter 5.

2.3.1 Information Criteria

Information criteria are used in time series analysis to select the best model among a set of candidates. They provide a balance between models complexity and goodness of fit, penalizing models with more parameters to avoid overfitting. Both criteria aim to guide the selection of models that are well-suited to the data yet remain as simple as possible, helping in choosing the model that is likely to be the best approximation of the underlying truth [12].

Akaike Information Criterion (AIC)

AIC evaluates the trade-off between the model fit to the data and the number of parameters used.

$$AIC = -2log(L) + 2k, \tag{2.1}$$

where L is the likelihood of the data and k is the number of parameters [12]. A lower AIC value indicates a better model, with a preference for models with fewer parameters, unless the additional complexity significantly improves the fit.

Bayesian Information Criterion (BIC)

Similar to AIC, BIC, also known as the Schwarz Criterion, adds a penalty term for the number of parameters in the model.

$$BIC = AIC + k[log(T) - 2], \tag{2.2}$$

where k is the number of parameters and T is the is the number of observations used for estimation [12].

Again, the objective of minimizing the BIC is to identify the optimal model. The model selected by BIC will be either identical to the one chosen by AIC or a simpler one with fewer parameters [12]. This criterion tends to penalize complexity more heavily than AIC.

2.3.2 Information Criteria Usage

When using Information Criteria for model selection, the process involves fitting multiple models to data and calculating the AIC or BIC for each model. Both AIC and BIC quantify the information loss when using a particular model to represent the process that generated the data. A lower value of AIC or BIC indicates a better model. However, while both criteria aim to penalize complexity to prevent overfitting, BIC tends to penalize complexity more heavily than AIC [4].

The process of selecting the best models consist of following steps:

- Fit each candidate model to data,
- Calculate the AIC or BIC for each model,
- Choose the model with the lowest AIC or BIC value [4].

2.3.3 Comparison of the models

The comparative evaluation of models is a critical step to ascertain their effectiveness and accuracy. When comparing time series models, following metrics can be used individually or in combination to provide a view of each model predictive capabilities. A lower value in these metrics generally indicates a better fit of the model to the observed data [27].

Root Mean Squared Error (RMSE)

RMSE is a standard measure that quantifies the square root of the average squared differences between the predicted and actual values. It is particularly sensitive to large errors. Root Mean Squared Error is given as:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n}} \tag{2.3}$$

where y_i denotes the actual value and $\hat{y}_i$ denotes the predicted value [27].

A relatively high weight is given to large errors by RMSE. This means that if the dataset contains large errors, a significantly higher RMSE will be observed, indicating poor model performance. It proves useful when larger errors are desired to be penalized more severely.

Mean Absolute Error (MAE)

MAE measures the average magnitude of errors in a set of predictions, without considering their direction.

$$MAE = \frac{1}{n}\sum_{i=1}^{n} \mid y_i - \hat{y}_i \mid \tag{2.4}$$

where y_i refers to the actual value and $\hat{y}_i$ represents the predicted value [27].

A straightforward measure of average error magnitude is provided by MAE without considering its direction. Unlike RMSE, the errors are not squared by MAE, meaning all errors are treated the same, which makes it less sensitive to outliers than RMSE.

Mean Absolute Percentage Error (MAPE)

MAPE expresses the error as a percentage, making it a useful measure for comparing the accuracy of models across different scales or datasets.

$$MAPE = \frac{1}{n}\sum_{i=0}^{n-1} \frac{|y_i - \hat{y}_i|}{max(\epsilon, |y_i|)}, \tag{2.5}$$

where y_i is the actual value, $\hat{y}_i$ denotes the predicted value and ϵ is an arbitrary small yet strictly positive number to avoid undefined results when y is zero *statsmodels* [1].

A clear interpretation in terms of percentage errors is provided by MAPE, which can be very intuitive for understanding the accuracy of predictions relative to true values. It is found to be especially useful when an interest in the relative error size is present.

[1]https://scikit-learn.org/stable/modules/model_evaluation.html#mean-absolute-percentage-error

Symmetric Mean Absolute Percentage Error (SMAPE)

SMAPE is an adjustment of MAPE that overcomes its asymmetry, offering a more balanced measure by considering both the forecast and the actual value in its calculation.

$$SMAPE = \frac{100}{n} \sum_{t=1}^{n} \frac{|F_t - A_t|}{(|A_t| + |F_t|)/2} \tag{2.6}$$

where A_t is the actual value and F_t is the predicted value [24].

SMAPE provides a symmetric measure of the accuracy of predictions, treating over-predictions and under-predictions equally. It is less affected by the problems associated with actual values close to zero, unlike MAPE.

2.4 Summary

Basic concepts of the time series data and analysis like plots, stationarity and correlation were introduced. This foundation paves the way for advanced analysis and modeling techniques. The following chapters will cover the full application of autocorrelation and partial autocorrelation plots, the stationarity test (ADF) and time series decomposition techniques applied to the selected dataset to extract and examine its underlying temporal structure and dynamics.

This chapter also covered the Information Criteria, which will be used in Chapter 5 to determine the number of model parameters, along with selected error metrics, which will provide a comparison of how well the models are performing in Chapters 5.

Chapter 3

Models for time series forecasting

This chapter is dedicated to the exploration of various models used for forecasting time series data. The Moving Average (MA) model is first introduced, recognized for its utility in smoothing out short-term fluctuations to emphasize longer-term trends. It is followed by introduction of Autoregressive (AR) models, where future values are depicted as being linearly dependent on their predecessors.

Additionally, the Autoregressive Integrated Moving Average (ARIMA) model is discussed, which merges the principles of AR and MA models and includes differencing to address non-stationary data effectively.

In addition to traditional statistical methods, this chapter explores the integration of machine learning techniques into time series forecasting including machine learning models like Long Short-Term Memory (LSTM) networks and Convolutional Neural Networks (CNNs). These models are highlighted for their capability to capture long-term dependencies in sequential data.

3.1 Moving Average model

Moving Average (MA) models are essential to time series analysis because they offer a straightforward but reliable way to project future values by using historical data.

In time series analysis, moving averages are used in forecasting models as well as for trend removal. The moving average detrends data by smoothing out short-term fluctuations to emphasize long-term trends. In other words, it reduces or eliminates the trend component, thereby preparing the data for additional analysis. In exploratory data analysis, this method is essential for identifying underlying patterns. On the other hand, the Moving Average (MA) model, which is frequently integrated into ARIMA models, is a more intricate statistical technique when used in forecasting. Here, it uses past forecast errors in a regression-like manner to predict future values, providing a sophisticated method to understand the influence of historical data on future predictions. While both applications (moving averages are used in forecasting models as well as for trend removal) leverage the concept of averaging past data, their objectives and implementations are distinct: one focuses on data preprocessing and pattern recognition, and the other on utilizing past errors for making informed future forecasts.

The *order* of a model, often denoted as $MA(q)$, is a crucial parameter that specifies the number of lagged forecast errors included in the model. Essentially, the order determines how many past error terms the model will use to predict the current value in a time series.

For instance, an MA(1) model uses just one previous error term, while an MA(2) model includes the two most recent error terms, and so on. The appropriate choice of the order is vital for the model accuracy. It should be high enough to capture the relevant patterns in the data but not so high that it introduces unnecessary complexity or overfitting.

MA model of order q is expressed as:

$$y_t = \mu + e_t + \theta_1 e_{t-1} + ... + \theta_q e_{t-q} \tag{3.1}$$

where:

- μ represents the mean of the data,
- e_t is the white noise,
- e_{t-n} are the forecasting errors,
- θ_n are the MA parameters [19].

Determining the optimal order of an MA model typically involves statistical analysis and model selection criteria, such as the Akaike Information Criterion (AIC) or the Bayesian Information Criterion (BIC), to balance the fit of the model with its simplicity [19].

3.2 Autoregressive model

Autoregressive models (AR) represent a fundamental category of linear models prevalent in time series analysis. Within these models, it is posited that the future values of a series are linearly dependent on a set number of its historical observations. These models, represented as AR(p), with p indicating the number of lag terms or the *order* of the model, essentially hypothesize that there exists a sequential dependency within the time series data. This dependency suggests that the information captured in previous time periods is useful in predicting future outcomes. AR(p) is described as:

$$y_t = \phi_0 + \phi_1 y_{t-1} + \phi_2 y_{t-2} + ... + \phi_p y_{t-p} + e_t \tag{3.2}$$

where:

- y_{t-n} are the previous values of the time series,
- ϕ_0 is the constant term,
- ϕ_n are the parameters of the model and
- e_t represents the error or the part of the current value that cannot be explained by the lagged terms [9].

Akaike Information Criterion (AIC) or the Bayesian Information Criterion (BIC) can be employed again to obtain optimal order of the AR model.

3.3 ARIMA model

Autoregressive Integrated Moving Average (ARIMA) models combine the principles of Autoregression (AR) and Moving Averages (MA) and incorporate differencing to ensure stationarity, which is reflected in the *integrated* part of their name. They are particularly powerful in handling non-stationary data where trends and seasonality are present, by integrating the past values (AR), past errors (MA), and differences of the observations to capture a broad range of time series behaviors. Flexibility to model various types of data makes ARIMA a popular choice for analysts in economic forecasting, stock market analysis, and many other fields.

As per [12], the form of an ARIMA(p,d,q) model can be written as:

$$y_t = c + \phi_1 y_{t-1} + ... + \phi_p y_{t-p} + \theta_1 e_{t-1} + ... + \theta_q e_{t-q} + \epsilon_t \tag{3.3}$$

where:

- c represents the constant term,
- ϕ_n are the parameters of the Autoregressive part of the model,
- y_{t-n} corresponds to the previous values of the time series,
- θ_n are the parameters of the Moving Average,
- e_{t-n} are the forecasting errors and
- ϵ_t is the white noise.

The Backshift notation, denoted as B, plays a crucial role in simplifying the representation of ARIMA (AutoRegressive Integrated Moving Average) models. In an ARIMA model, the backshift notation is used to express the process of shifting a time series back by a specific number of periods. For instance, if y_t represents the current value of the series at time t, then:

$$B(y_t) = y_{t-1} \tag{3.4}$$

represents the value of the time series at time $t - 1$ [12].

Using the backshift notation, ARIMA(p,d,q) model can be written as:

$$(1 - \phi_1 B - ... - \phi_p B^p) \times (1 - B)^d y_t = c + (1 + \theta_1 B + ... + \theta_q B^q)\epsilon_t \tag{3.5}$$

where:

- $(1 - \phi_1 B - ... - \phi_p B^p)$ repsents the *AR(p)* term,
- phi_n are the parameters of the autoregression,
- $(1 - B)^d y_t$ corresponds to d differences,
- $c + (1 + \theta_1 B + ... + \theta_q B^q)\epsilon_t$ is the *MA(q)* term,
- θ_n are the parameters of the Moving Average,
- c is the constant term and
- ϵ_t is the white noise [12].

3.4 Parameters estimation

After determining the model order, it becomes necessary to calculate the parameters θ_n and ϕ_n. For fitting AR models, *statsmodels* [1] library uses Ordinary Least Squares method (OLS) [23]. OLS regression is a method used to optimize the fit of a straight line within a linear regression framework, ensuring it aligns as closely as possible with the given data points. It is widely regarded as a highly effective optimization technique for linear models because it offers unbiased estimations for the intercept and slope coefficients in the regression equation [2].

MA and ARIMA models are fitted using state space and Maximum likelihood estimation via the Kalman filter, which is a statistical method used to estimate the parameters of a model, which is especially useful in the context of time series data that can be described by state space models. State space models [16] are a framework for modeling time series data that consist of two main components: an observation equation that describes the relationship between the observed data and the unobserved state variables, and a state equation that explains how the state variables evolve over time [22].

3.5 Convolutional Neural Network (CNN)

Originally developed for image processing tasks, Convolutional Neural Networks (CNNs) have expanded their utility to sequential data analysis, such as time series forecasting. Their hierarchical feature extraction capabilities enable them to adeptly navigate and identify the intricate temporal patterns that are characteristic of time series datasets [15].

CNNs excel in time series forecasting due to their local connectivity, allowing for effective short-term pattern recognition, and hierarchical representation capabilities, enabling the capture of both short and long-term dependencies. Additionally, their ability to process multiple sequences simultaneously enhances efficiency, especially with large or multivariate datasets. This blend of features positions CNNs as a robust tool for analyzing temporal data [15].

Convolutional Neural Networks (CNNs) have transformed the landscape of time series forecasting with their ability to learn complex patterns. However, their application comes with limitations that require strategic consideration. The fixed input length requirement necessitates preprocessing steps that could lead to data loss. While adept at identifying local patterns, CNNs may falter in capturing long-term dependencies, a gap potentially filled by Recurrent neural networks (RNNs) or attention mechanisms. The nature of CNNs poses challenges in interpreting their decision-making processes. Their reliance on extensive datasets and computational resources, along with a deterministic output that lacks uncertainty estimation, further complicates their use in environments where data is scarce, hardware is limited, or probabilistic forecasts are crucial. These constraints highlight the importance of carefully assessing the suitability of CNNs for specific forecasting tasks, considering alternative models and strategies to address their shortcomings [15].

3.5.1 Convolution

Convolution involves sliding a kernel, which is a smaller array, across a larger one to create a new matrix. At every position, the kernel values are multiplied by the values of the corresponding subsection of the larger matrix and summed up, producing one element of

[1]https://www.statsmodels.org/stable/index.html

the new matrix. This sliding and multiplying action is performed multiple times with different kernels to highlight various features of the original matrix [19].

Traditional convolution techniques, commonly suited to image analysis due to their space-invariant nature, often fall short when applied to time series data. In time series analysis, the temporal proximity and scale of data points are crucial, with more recent data often being more relevant than older data. Moreover, scale invariance, a beneficial trait for identifying objects in images regardless of size, may overlook important scale-dependent patterns in time series, such as distinguishing between daily and yearly fluctuations [19].

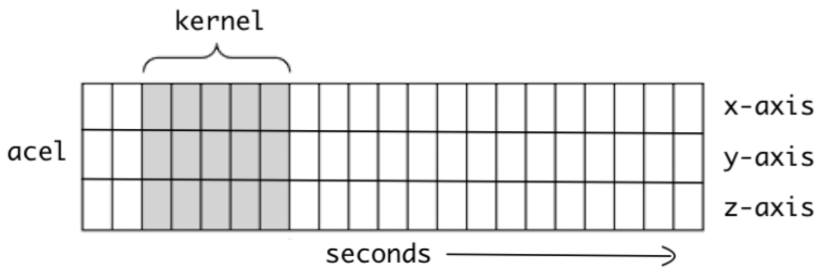

Figure 3.1: Visual representation of 1D convolutional kernel [25].

Figure 3.1 shows graphical representation of a 1D convolutional kernel applied to a three-dimensional input. This 1D convolution process is typically used in signal processing or for analyzing any form of temporal or sequential data, such as audio signals, time series data.

3.6 Recurrent networks

Recurrent neural networks (RNNs) are a type of neural network specially designed for sequential data, repeatedly applying the same weights to each input over time. Despite their similarity to feedforward neural networks, RNNs excel in tasks involving sequences, such as language processing, forecasting, and time series classification, because they can maintain information across inputs due to their internal memory. Some of the key differences between feed forward and RNNs are:

- RNNs process data in a time-ordered sequence.
- RNNs carry a state from one time step to the next, influencing responses to new inputs.
- Parameters within RNNs are designed to update this state, including the hidden state, as they progress through time steps [8].

The figure 3.2 depicts a simplified representation of a Recurrent Neural Network (RNN) cell at a single time step. It illustrates the key components of the RNN: the input x_t at the current time step, the hidden state h_t which is passed to the next time step, and the recurrent connection that loops the hidden state back into the cell. The cell block labeled A represents the computation that happens at each time step, taking into account both the current input and the previous hidden state [20].

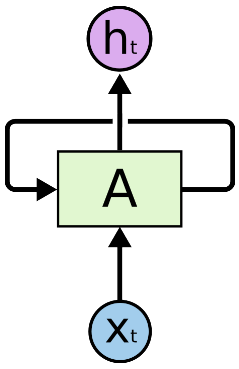

Figure 3.2: Recurrent cell [20].

3.6.1 Gated recurrent unit (GRU)

The Gated Recurrent Unit (GRU) is a type of artificial neural network architecture that is widely used in the field of deep learning, particularly in the processing of sequential data. GRUs are an evolution of the traditional recurrent neural network (RNN) model designed to better capture dependencies for sequences of data over long durations. They achieve this through a specialized structure that includes gating mechanisms to regulate the flow of information [6].

GRUs incorporate two key gates: the update gate and the reset gate. The update gate helps the model to decide the amount of past information (from previous time steps) that needs to be passed along to the future. Meanwhile, the reset gate determines how much of the past information to forget. This gating mechanism addresses the vanishing gradient problem that is common in standard RNNs, making GRUs capable of learning dependencies from long input sequences without losing significant information over time [8].

What sets GRUs apart from other sequence processing models, such as Long Short-Term Memory (LSTM) units, is their simplified structure which uses fewer parameters. This makes them generally faster to train, without a substantial compromise on performance for many tasks. GRUs have been successfully applied in various domains including natural language processing (NLP), speech recognition, and time series analysis [8].

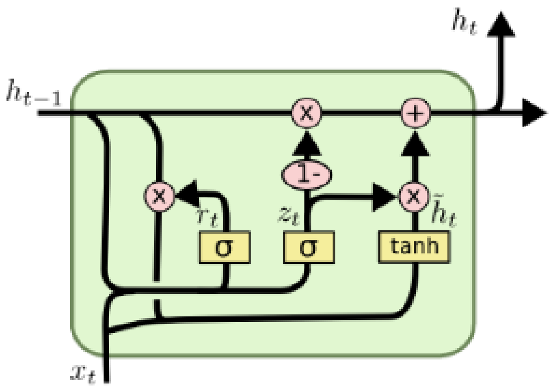

Figure 3.3: GRU cell [20].

Figure 3.3 illustrates the internal structure of a Gated Recurrent Unit (GRU) cell. The GRU cell is designed to process sequential data and is capable of remembering and forgetting information dynamically through its gates. Inputs to the GRU cell are:

- h_{t-1} represents the hidden state from the previous time step, carrying information from earlier in the sequence,
- x_t is the input at the current time step.

Inside the GRU cell, there are following components:

- Update gate (z_t) this gate decides the extent to which the previous hidden state h_{t-1} is carried to the current hidden state h_t,
- Reset gate (r_t) determines how much of the past information to forget,
- Candidate hidden state ($\widetilde{h_t}$) is the result of combining the current memory content with the reset information. It is a combination of the new input and the information allowed through by the reset gate [8].

The gates use the sigmoid activation function σ to output values between 0 and 1, indicating the proportion of information to pass through. The actual update to the hidden state is done through a weighted sum, where the update gate allows parts of the old state h_{t-1} to be *updated* with the new candidate hidden state $\widetilde{h_t}$ [8].

3.6.2 Long Short-Term Memory (LSTM)

Long Short-Term Memory (LSTM) is a recurrent neural network architecture introduced to overcome the limitations of traditional recurrent networks in learning to store information over extended time intervals. Traditional methods struggle with long-term dependencies due to vanishing or exploding gradients. LSTM addresses this challenge by introducing a mechanism to maintain constant error flow across long sequences, allowing it to learn dependencies that exceed 1000 discrete time steps without performance degradation [10].

The core innovation in LSTM is the use of special units called „constant error carrousels“ (CECs), which enable constant error propagation through time by restricting the flow of information. These units are augmented with multiplicative gate units, input gates, output gates, and forget gates. These gates control the flow of data into and out of the CECs and allow the LSTM to selectively remember or forget information, making it highly effective for tasks that require learning from long-term dependencies [10].

LSTM networks are local in space and time, meaning that their computational complexity per time step and weight is constant, making them efficient and scalable. They demonstrate superior performance on a range of artificial tasks involving long time lags and complex dependencies, outperforming traditional recurrent networks and other contemporary methods in terms of learning speed and success rate [10].

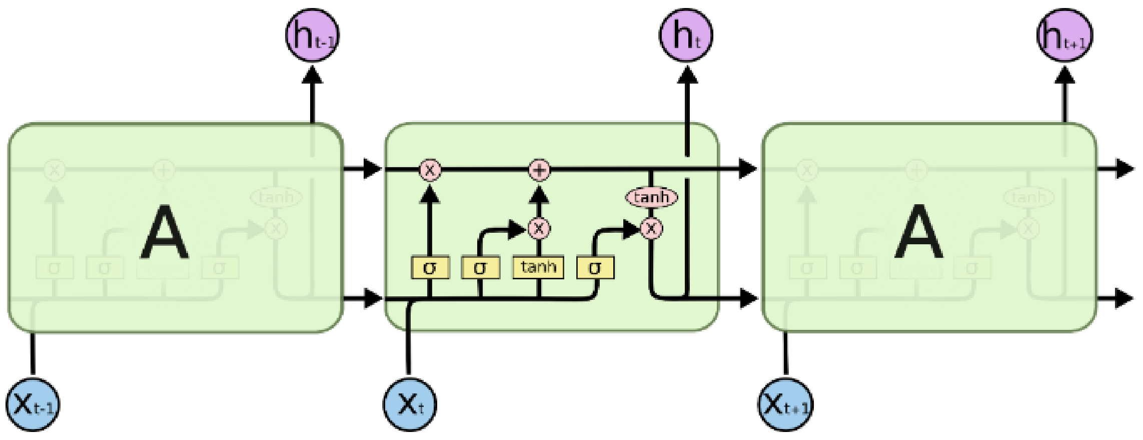

Figure 3.4: LSTM cell [20].

Figure 3.4 depicts a typical Long Short-Term Memory (LSTM) unit for three time steps in a recurrent neural network. The LSTM unit is designed to allow information to be retained or discarded through gates that regulate the flow of information. The LSTM cell consists of:

- Input (X_t): The input vector at time step t.
- Hidden State (h_t): The hidden state vector at time step t, which is passed along to the next time step.
- Cell State Line (horizontal line across the top): Represents the cell state that is passed through the sequence, with modifications governed by gates within the LSTM.
- Sigmoid Functions (σ): Represented by circles with σ, these are activation functions used within the input, forget, and output gates. They output values between 0 and 1, effectively controlling the extent to which information is allowed to pass through the gates.
- Tanh Function: Represented by the tanh label inside the cell, this activation function outputs values between -1 and 1 and is applied to the cell state and combined with the input, and also to generate the output.
- Pointwise Operations: The multiplication ($\times$) and addition ($+$) signs indicate point-wise operations between vectors, such as when the input gate determines how much new information to add to the cell state or when the forget gate scales the existing cell state.

The arrows depict how data flows through the LSTM unit: from the input to the output and across to the next cell in the sequence. This includes the flow of the cell state across the top, which is the key to the ability of the LSTM to maintain memory over time [5].

3.7 Parameter estimation

Parameter estimation in neural networks is a complex and necessary method for creating good deep learning models. The goal is to identify the best collection of weights and biases that minimize prediction error, allowing the network to successfully generalize from training data to new data. This procedure is primarily reliant on advanced optimization algorithms that update parameters depending on the gradients of a loss function [8].

Depending on their architecture, neural networks can have millions of parameters. These parameters are modified iteratively during training, generally using *backpropagation* in conjunction with an optimization algorithm such as *Stochastic Gradient Descent (SGD)* or its variations *Adam* or *RMSprop*. These techniques change the parameters based on the gradients determined during backpropagation, indicating which direction the parameters should be altered to reduce loss [8].

Hyperparameter tuning is also essential for parameter estimation. In contrast to model parameters, hyperparameters are established prior to training and impact the learning process rather than being learned directly from data. These include the learning rate, batch size, epoch count, and architecture-specific parameters like as the number of layers and hidden units. Extensive testing is sometimes required to tune hyperparameters, which can have a major impact on model performance. Grid search, random search, and Bayesian optimization are prominent techniques for efficiently exploring the hyperparameter space [8].

3.8 Summary

This chapter explores various models used for forecasting time series data, which are essential for the efficient operation of network communication systems. Initially, traditional statistical models are introduced. Among them, the Moving Average (MA), Autoregressive (AR), and Autoregressive Integrated Moving Average (ARIMA) models are emphasized. These models are recognized for their effectiveness in capturing linear relationships within time series data.

More sophisticated machine learning techniques are introduced. Long Short-Term Memory (LSTM) networks and Convolutional Neural Networks (CNNs) are examined for their ability to detect complex, non-linear patterns in time-dependent data, offering substantial improvements over traditional approaches, especially in managing large datasets and capturing long-term dependencies.

Chapter 4

Dataset description and analysis

Given the proprietary nature of AT&T datasets, a publicly available dataset from Kaggle was opted for this thesis. This approach is ensured to comply with data privacy standards and allows for broader applicability and reproducibility. For this reason, the dataset *Network Analytics* [21] was selected. This chapter is dedicated to describing this dataset and its statistical properties, which is essential to the practical use of the previously described methods. This dataset, which is publicly available, was obtained from Kaggle public domain, a website that is popular for having a vast collection of datasets.

4.1 Data overview

Timestamp	OutboundUtilization (%)
10/10/2017 7:01	96.2442
10/10/2017 7:21	99.1131
10/10/2017 7:26	97.2892
10/10/2017 7:31	98.0286
10/10/2017 7:36	99.5263

Table 4.1: First 5 rows of the dataset.

This dataset consists of one feature, *Outbound Utilization (%)*. The first column, *Timestamp*, records the precise moments at which the corresponding data points were captured, thereby anchoring each observation in time. The second column, *Outbound Utilization (%)* quantifies the utilization levels of a networking device expressed as a percentage. This column is pivotal as it provides a numerical representation of the device operational capacity at each recorded timestamp.

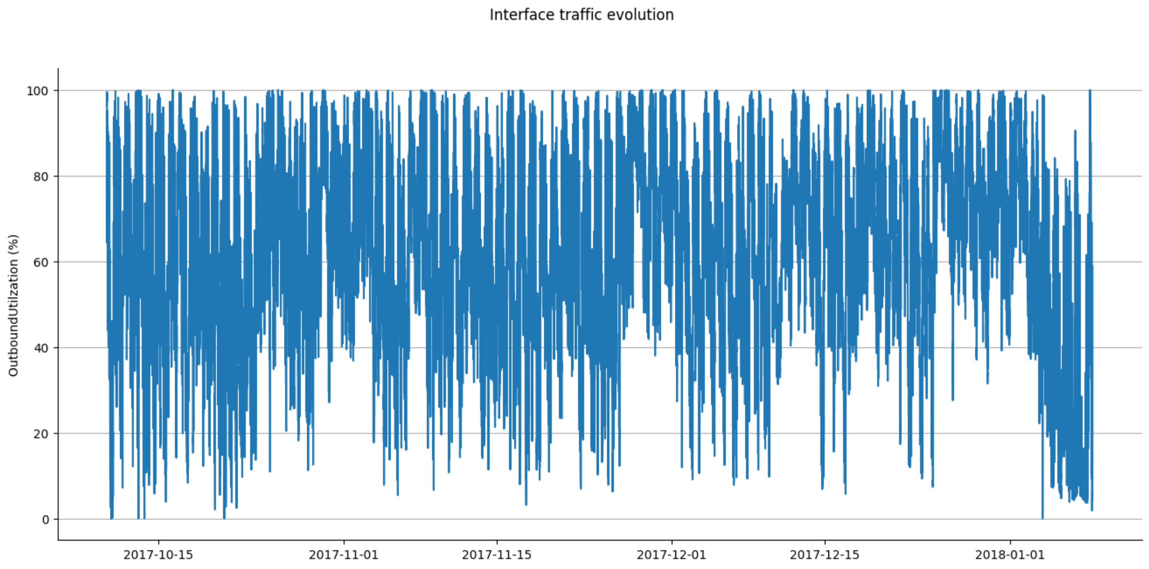

Figure 4.1: Graph of the Network Analytics dataset.

Figure 4.1 provides the visual representation of the data. The data points are showing fluctuations, suggesting that the interface occupation is highly variable, but also regularly reaches full capacity. There is a consistent pattern where the occupation percentage spikes to near 100% and drops back down, which might indicate recurring high-traffic events or possibly automated processes that generate traffic at regular intervals.

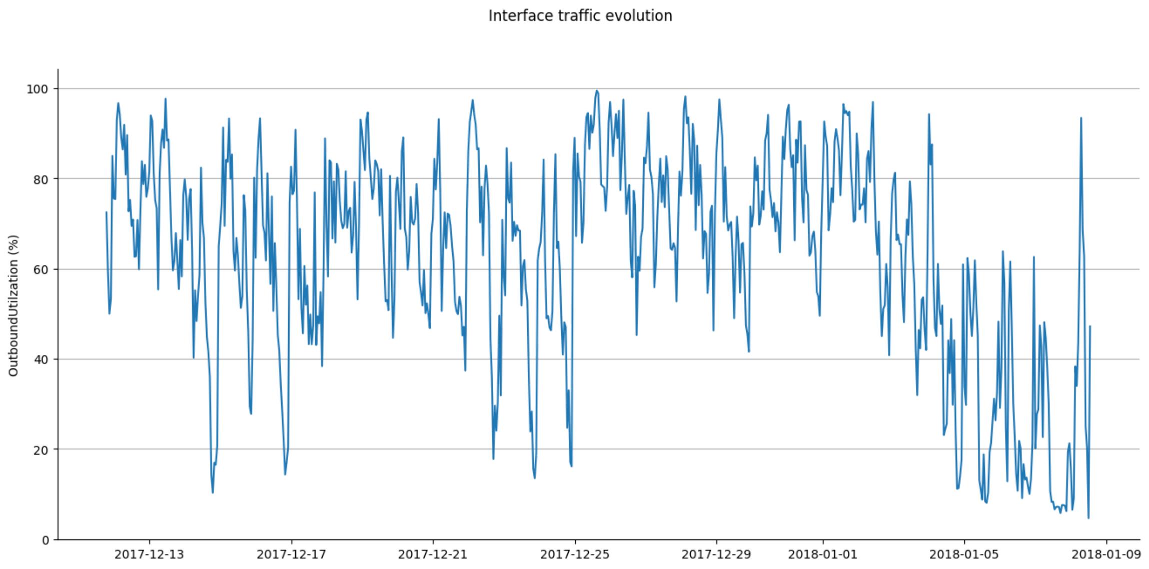

Figure 4.2: Graph of the Network Analytics dataset resampled.

To get better understanding of the data, figure 4.2 shows the dataset resampled to one hour intervals and mean of each such interval was taken. There are noticeable dips, possibly suggesting periods of low traffic or downtime, and peaks which indicate moments of high occupation. The overall variability seems significant, and there doesn't appear to be a clear, regular pattern. The traffic seems sporadic and unpredictable, with some very sharp spikes and drops.

4.2 Data characteristics

count	25 631
mean	60.372335
std	22.517275
min	0.000048
$Q_{0.25}$	44.873850
$Q_{0.50}$	62.100200
$Q_{0.75}$	77.702150
max	99.995500

Table 4.2: Network Analytics dataset description.

Table 4.2 holds some basic information about the dataset. The dataset consists of 25 631 samples. The mean outbound utilization is approximately 60.37% indicating an average operating level just above the half of its capacity. The data exhibits a considerable spread as evidenced by a standard deviation of about 22.52%, which points to significant variability in utilization rates. The range of utilization is vast, stretching from minimum of nearly 0% to maximum of almost 100% (99.9955%), underscoring periods of both minimal and maximal usage. The utilization values exhibit a slight skew towards higher ones as indicated by the 25th percentile of 44.87% and the median, or 50th percentile, being slightly higher at 62.10% than the mean. On the other hand, the top quarter ofp data points show relatively high utilization rates, as indicated by the 75th percentile, which is at 77.70%.

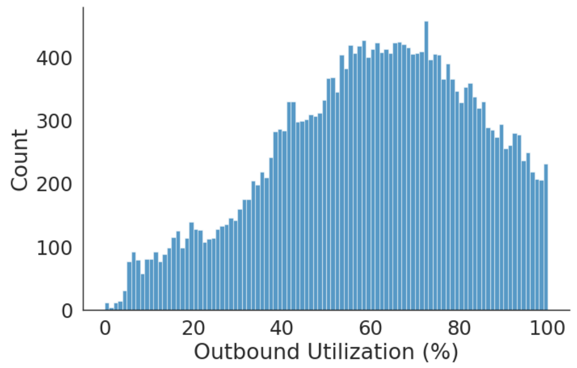

Figure 4.3: Outbound Utilization (%) distribution.

Figure 4.3 presents a detailed histogram of the data frequency distribution. The bars represent counts of a measurement at different utilization percentages. The bars increase in height from 0% utilization, peak around the 60-80% utilization range, and then decrease in height toward 100% utilization. This pattern suggests a distribution where the most common utilization percentages are around the middle to higher end of the scale, but not often at full capacity (100%).

Leveraging the capabilities of the Python *fitter* [7] library, which facilitates the comparison of data across a multitude of potential distributions, it was determined that the beta distribution [13] provides the most accurate model for the dataset. Figure 4.4 illustrates the fitted distribution curve, while Table 4.3 contains parameters of the fitted distribution.

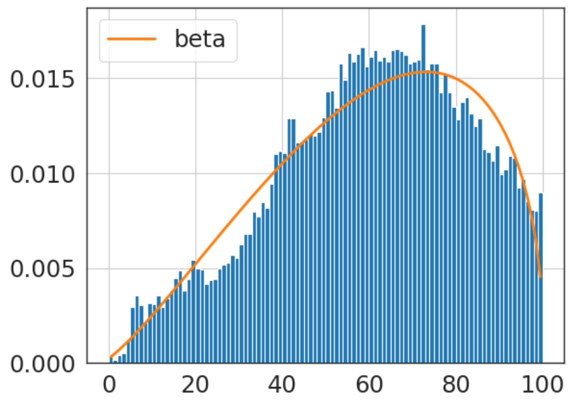

Figure 4.4: Fitted distribution.

a	2.2840246378455658
b	1.4660546616390748

Table 4.3: Parameters of the fitted beta distribution.

To confirm the hypothesis that the dataset conforms to the identified fitted distribution, the Python *scipy* [26] library provides an implementation of the Kolmogorov-Smirnov test [17].

Test statistics	0.021276993681405676
p-value	1.6418667607936974e-10

Table 4.4: Results of the Kolmogorov-Smirnov test.

Table 4.4 shows the result of the Kolmogorov-Smirnov test. The test statistic (around 0.02127) is relatively small, which suggests that the difference between the empirical distribution function of the sample and the cumulative distribution function of the comparison distribution may not be large. However, the p-value is very small (in the order of 10^{-10}), which typically indicates that the results are statistically significant and that the null hypothesis (that the sample comes from the specified distribution) can be rejected.

4.2.1 Autocorrelation

Autocorrelation, which is described more in depth in Chapter 2, is a statistical measure that expresses the degree of correlation between a time series and a lagged version of itself over successive time intervals. It quantifies how much the current value of the series is related to its past values, providing insights into the repeating patterns or seasonal effects.

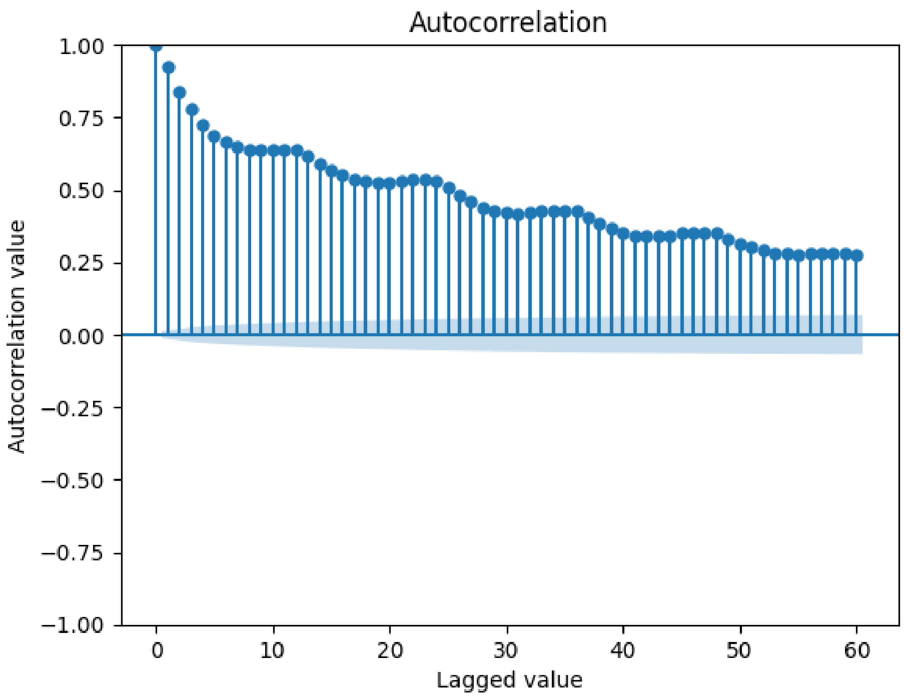

Figure 4.5: Autocorrelation plot of the Network Analitycs dataset.

Image 4.5 presents a autocorrelation plot, which graphically demonstrates how much is the current value of the time series influenced by its previous values. The plot shows a slow decrease, which implies that the time series does not remember its past values as strongly over time, and the connection between past and present data gets weaker the more apart its values are.

4.2.2 Partial autocorrelation

Partial autocorrelation measures the direct relationship between an observation and its lagged version at a specific interval, discounting the influence of correlations at intermediate lags.

Figure 4.6 shows the partial autocorrelation plot, which is used to visualize and measure the degree of association between a time series and its lagged values, independent of intermediate lags. After the first lag, the partial autocorrelation values quickly drop near zero and fluctuate slightly above and below the zero line for the remaining lags, suggesting that there is little to no partial autocorrelation at these lags once the effects of shorter lags have been accounted for. This pattern is often indicative of an AR(1) process in time series data, where *AR* stands for *autoregressive* and the *(1)* indicates that the only significant autocorrelation is at lag 1. This could mean that the time series data can be effectively modeled using an AR(1) model [14].

4.2.3 Stationarity

Stationarity in time series data refers to a condition where key statistical properties of the dataset, specifically the mean, variance, and autocorrelation, remain constant over time.

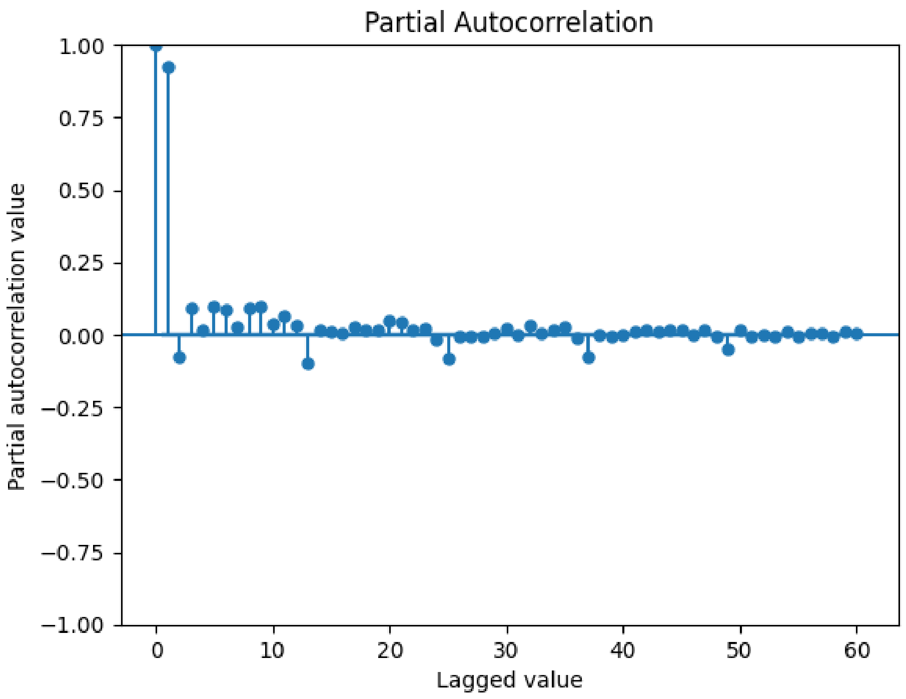

Figure 4.6: Partial autocorrelation plot of the Network Analitycs dataset.

Measurement of the stationarity of the data was done using Augmented Dickey-Fuller, from the python module statmodels [1].

Test statistics	-12.704162687630898
p-value	1.0639878489318627e-23

Table 4.5: Results of the Augmented Dickey-Fuller test.

Table 4.5 displays results from an Augmented Dickey-Fuller test, indicating a test statistic of approximately -12.70 and an extremely small p-value of about 1.06e-23. These results strongly suggest that the time series is stationary, meaning its statistical properties do not depend on the time at which the series is observed. Given the highly negative test statistic and the p-value being practically zero, the null hypothesis of a unit root (indicating non-stationarity) can be confidently rejected. This implies that the time series data likely do not have a trend or seasonal effect.

4.3 Summary

This chapter delved into statistical descriptions and visual representations of the Network Analytics dataset, revealing fluctuating utilization levels with spikes and dips suggesting varying network traffic. Key statistical measures like mean, standard deviation, and maximum values are provided, indicating significant variability in network usage. Autocorrelation and partial autocorrelation were explored, suggesting that the time series might be modeled effectively using an AR(1) model. The Augmented Dickey-Fuller test is applied, leading to the conclusion that the time series is stationary—crucial information for subsequent modeling.

This chapter sets the groundwork for applying the forecasting methods by providing a detailed understanding of the dataset's characteristics, crucial for effective implementation of the models described in chapters 3 and implemented in chapter 5.

Chapter 5

Implementation

This chapter implements the theoretical concepts introduced in earlier chapters. Initialy, statistical models are applied to the dataset. The implementation details, including model selection and parameter tuning, are discussed. Following the application of statistical models, the implementation of more sophisticated machine learning techniques, including Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) such as GRU and LSTM, is examined.

5.1 Statistical models implementation

The Python library *statsmodels* [1] provides a suite of tools for statistical modeling, including robust implementations of the Moving Average (MA), Autoregressive (AR), and Autoregressive Integrated Moving Average (ARIMA) models.

5.1.1 Moving Average model

Order selection

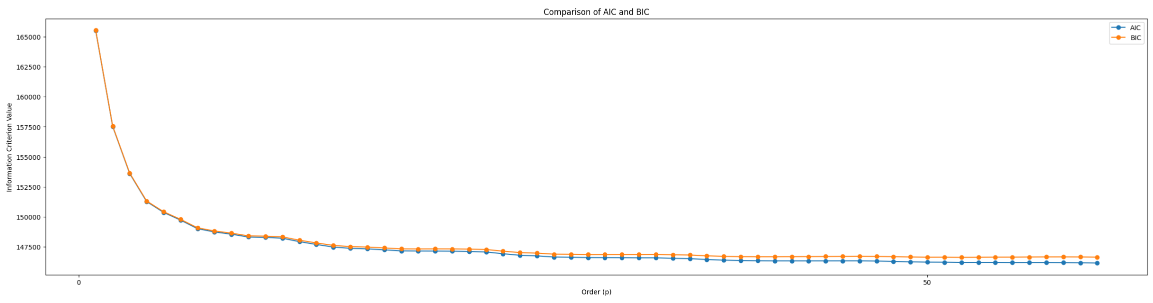

Figure 5.1: Evaluation of the Information criterion of the MA model.

The evolution of the Akaike Information Criterion (AIC) and Bayesian Information Criterion (BIC) with increasing order of the Moving Average model is depicted in the figure 5.1. Order of the model was determined based on the BIC value, chosen for its superior balance between model performance and complexity.

[1] https://www.statsmodels.org/stable/index.html

According to the insights from Chapter 2, the optimal models are identified through the minimization of either the Akaike Information Criterion (AIC) or the Bayesian Information Criterion (BIC). It has been observed that the lowest BIC value, which indicates the most suitable model in terms of balancing goodness-of-fit and model simplicity, corresponds to a model order of 52.

Implementation

The foundation for the Moving Average (MA) model analysis was set using the *statsmodels* [1] package. Specifically, by suitably configuring the *ARIMA* model available within this package, it can be transformed to function solely as an MA model. This adjustment ensures that the ARIMA model effectively simulates an MA model.

```
from statsmodels.tsa.arima.model import ARIMA

MA_model = ARIMA(endog=train, order=(0, 0, 52))
MA_model = MA_model.fit()
```

Listing 5.1: MA model definition

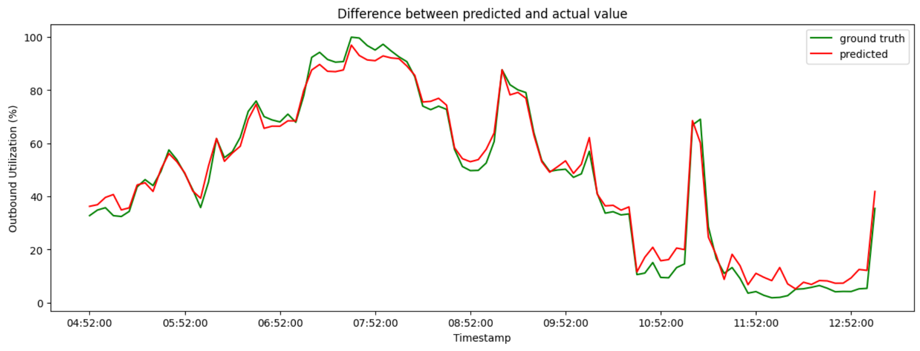

Figure 5.2: Difference between actual and predicted value of the MA model.

Figure 5.1 presents the difference between the actual value and value predicted by the Moving Average (MA) model.

- The predicted values closely follows actual values for most of the timeframe, which suggests that the moving average model is performing well in predicting those values.
- Both lines show similar peaks and troughs, which means the predictive model is capable of catching the trend and seasonality of the data.
- There are a couple of sharp spikes in both the predicted and actual values, particularly noticeable around 10:52:00. These could represent outliers or sudden unexpected changes in utilization. The model seems to predict these spikes, although with slightly less intensity, which suggests it may have some robustness against noise but may not be entirely sensitive to very abrupt changes.

[1] https://www.statsmodels.org/stable/index.html

- There are moments, such as shortly after 06:52:00 and 10:52:00, where the predicted values lag behind the actual values.
- The model's accuracy appears to decrease during the periods with the sharpest changes in utilization (both up and down). This is typical for moving average models, which tend to smooth out the data and might not be as responsive to sudden shifts.

5.1.2 Autoregressive model

Order selection

Drawing upon the findings from Chapter 4, the order of the Autoregressive (AR) model was established to be 1, signifying a single lag in the time series data as the optimal parameter.

Implementation

To set a foundational baseline for the Autoregressive (AR) model, the *AutoReg* class from the *statsmodels* library was employed. This specific class provides a robust framework for fitting AR models, offering a comprehensive suite of tools for model estimation and analysis, thereby ensuring a precise and effective establishment of the baseline AR model parameters.

```
from statsmodels.tsa.ar_model import AutoReg

model = AutoReg(train, lags=1)
model = model.fit()
```

Listing 5.2: AR model definition

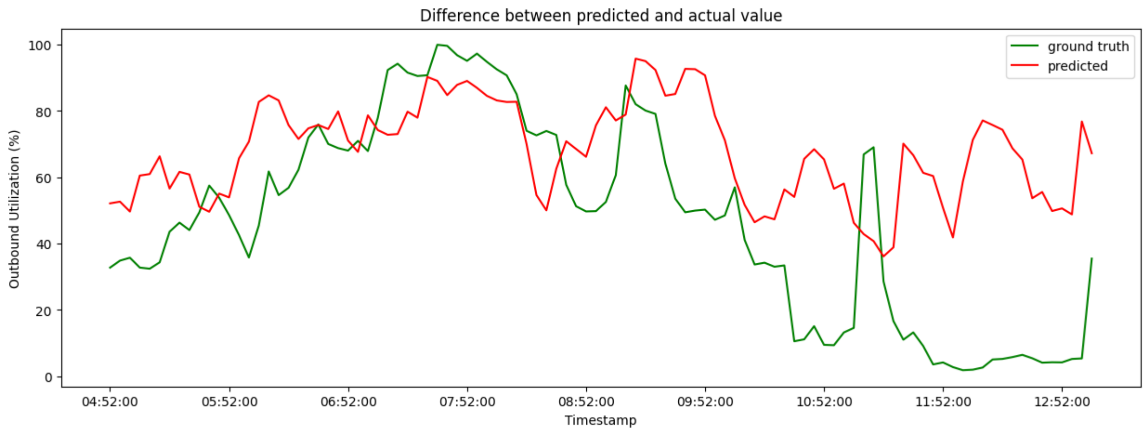

Figure 5.3: Difference between actual and predicted value of the AR model.

The accuracy of the predicted values in the figure 5.3 can be described as inconsistent. Some of the key outputs from this figure are:

- The predicted values seem to generally follow the overall trend of the actual values. This indicates that the autoregressive model captures the trend of the data.

- Compared to a moving average model, an autoregressive model should be better at reacting to recent changes because it uses previous data points in a more complex way. This plot suggests that the model is somewhat reactive to changes, but there are still notable discrepancies at certain points.
- There are instances where the predicted values diverge sharply from the actual values, particularly at around 08:52:00 and 10:52:00, where the predicted values overshoot or undershoot significantly. These points indicate moments where the model failed to accurately predict sudden changes.
- The autoregressive model appears to exhibit some volatility, with the predicted line showing more pronounced ups and downs compared to the actual line. This could be due to the model overfitting to the noise in the data or being overly sensitive to recent fluctuations.
- In some sections, the red line leads the green line, while in others, it lags behind. For example, shortly after 06:52:00, the predicted value precedes the actual value, suggesting that the model anticipated a change before it occurred. Conversely, around 08:52:00, the predicted value lags the actual value, indicating a delay in response.
- The magnitude of the errors (the vertical distance between the red and green lines) is quite variable. While sometimes the predicted values are very close to the actual values, at other times they are quite far, indicating inconsistent predictive performance.

5.1.3 ARIMA model

Order selection

The order of the ARIMA model was determined by integrating the insights derived from both the Moving Average (MA) and Autoregressive (AR) models.

Implementation

Due to the extensive training time required by the ARIMA model, an alternative approach was needed. The data was aggregated into mean values over 2-hour intervals, substantially downsizing the dataset. However, even with this modification, the model did not fully converged after a single application of the Maximum Likelihood estimation. This outcome suggests potential for further enhancement, indicating the need for additional fine-tuning of the parameters.

The initial parameters for the ARIMA model were determined based on the parameters of the MA and AR models. This approach resulted in the configuration of an ARIMA model with an order of (1, 0, 52).

```
from statsmodels.tsa.arima.model import ARIMA

model = ARIMA(train, order=(1, 0, 52))
model = model.fit()
```

Listing 5.3: ARIMA model definition

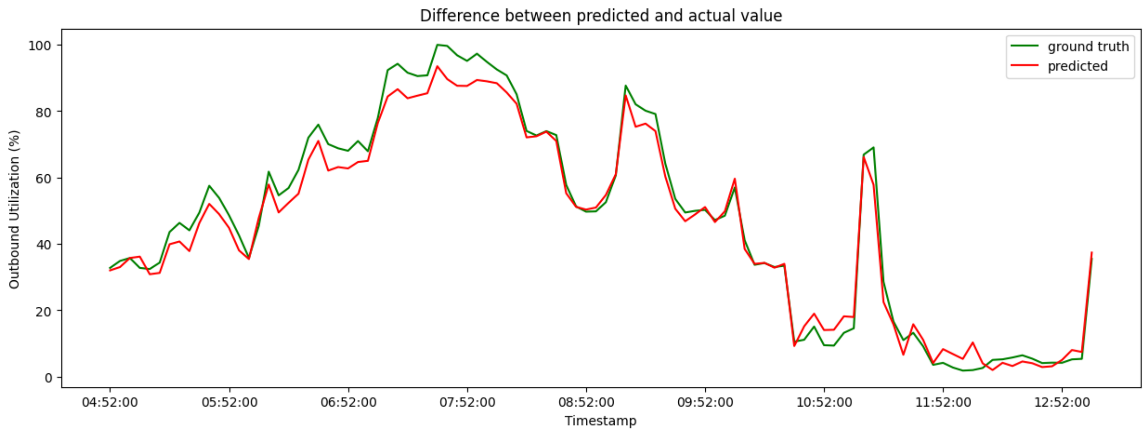

Figure 5.4: Difference between actual and predicted value of the ARIMA model.

Figure 5.4 represents the performance of the ARIMA model.

- Both the predicted and actual lines seem to follow the same general trend, which indicates that the ARIMA model has captured the overall pattern of the data.
- At many points, particularly from 04:52:00 to around 08:52:00, the predicted values closely match the actual values. However, there are intervals, such as around 09:52:00 and 11:52:00, where the prediction deviates significantly from the actual data, indicating periods of lower predictive accuracy.
- Around 10:52:00, there is a large spike in both the predicted and actual values, with the actual values showing a much sharper peak. This suggests that while the ARIMA model detected the anomaly, it underestimated its magnitude.
- The ARIMA model shows some ability to respond to volatility. However, the degree of response is inconsistent, as seen in the various peaks and trough.
- The model predictions fluctuate alongside the actual values, which is a positive sign of its responsiveness to changes. However, these fluctuations are not always synchronized, leading to noticeable differences at certain times.

5.2 Neural Networks implementation

The following neural network models were developed using the PyTorch library, employing the *Adam* optimizer and utilizing *Mean Squared Error* as the loss function.

5.2.1 Convolutional Neural Network (CNN)

The CNN architecture depicted in Figure 5.5 consists of the following layers and operations:

- Input Tensor: The network takes an input tensor with the shape (1, 5, 1), which implies a single sample with 5 channels, which is a value based on Figure 5.6, and a height/width of 1.
- First Convolutional Layer: This layer processes the input tensor with kernels that expand the depth from 5 channels to 48. The output shape after this layer is (1, 48, 1).

- First Activation Function (ReLU): The rectified linear activation function is applied element-wise, maintaining the shape (1, 48, 1).
- Second Convolutional Layer: Another convolutional layer takes the output from the previous ReLU layer and increases the depth from 48 to 96, resulting in an output shape of (1, 96, 1).
- Second Activation Function (ReLU): Again, a ReLU activation is used, leaving the shape unchanged at (1, 96, 1).
- View/Reshape Operation: This operation reshapes the output tensor from the second ReLU activation into a one-dimensional tensor with 96 elements.
- First Linear (Fully Connected) Layer: The reshaped tensor is passed through a fully connected layer, reducing its dimensionality from 96 to 20.
- Third Activation Function (ReLU): The ReLU activation function is applied to the output of the first fully connected layer, with the shape remaining (20,).
- Second Linear (Fully Connected) Layer: A second fully connected layer further reduces the dimensionality from 20 to 1.
- Output Tensor: The final output tensor of the network is a single value with the shape (1,).

The architecture of the network is based on architecture described in the article Convolutional Neural Networks for Energy Time Series Forecasting [1].

Hyperparameters

Convolutional Neural Networks (CNNs) feature a number of hyperparameters that control their architecture and training behavior. Among these is the input length, which is especially important for 1D CNNs that handle with sequences or time-series data. This sets the size of the one-dimensional input that the network expects, as well as the temporal or sequential length that each layer will examine.

Figure 5.6 offers an in-depth look of how changing the input length affects the model performance and allows for configuration selection. The fluctuations imply that the relationship between input length and model loss is not simple and may be impacted by additional variables not shown in the graph. It also implies that there may not be a single optimal input length but rather a range within which the model performs adequately.

Other important hyper parameters include the kernel size, which influences the receptive field of the filters; the number of filters per layer, which determines the ability to detect various features; the stride and padding, which affect the dimensionality of the output feature maps; and the activation functions, such as ReLU or sigmoid, which introduce non-linearities required for learning complex patterns. Furthermore, training hyper parameters like as batch size. These hyper parameters as well as the architecture of the model are based on paper published on IEEE Xplore [1].

[1]https://ieeexplore.ieee.org/document/8489399

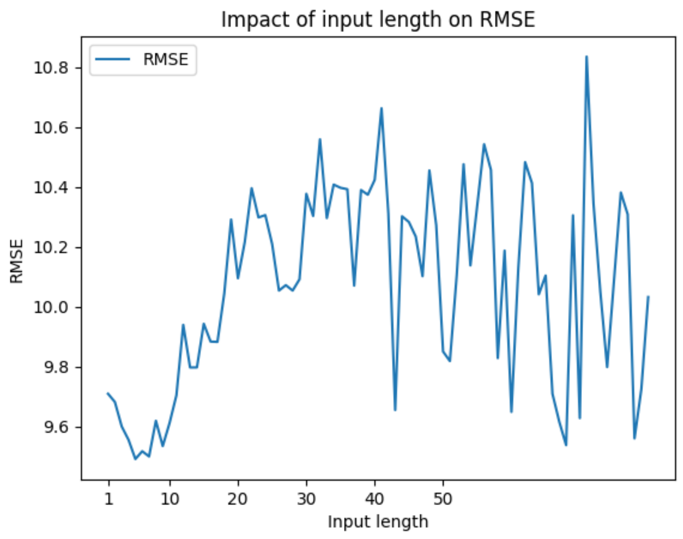

Figure 5.6: Dependence between CNN model input length and RMSE.

Input length	5
Kernel size	5
Batch size	16
Learning rate	0.0001
Training epochs	200

Table 5.1: Hyperparameters of the CNN model.

Table 5.1 shows the hyperparameters of the Convolutional Neural Network (CNN) model. These hyperparameters are:

- Input length: This hyperparameter defines the length of the input sequences fed into the model. Each sequence will be of length 5, which implies that the CNN will consider 5 time steps of data for each prediction it makes.
- Kernel size: This number refers to the size of the filter used in the convolutional layer, which will affect how the input data is processed. A size of 5 indicates that the filter covers five units at a time when applied to the input data.
- Batch size: This specifies the number of training samples that the model will process before updating the internal model parameters.
- Learning rate: The learning rate determines the size of the steps taken during gradient descent optimization. A smaller learning rate of 0.0001 means the model will make smaller, more precise updates to the model weights.
- Training epochs: This indicates the total number of times the learning algorithm will work through the entire training dataset.

Forecasting

The figure 5.7 presents the difference between ground truth and value predicted by CNN model.

- Both the ground truth and the predicted lines show a strong correlation in terms of the overall trend throughout the time series. This implies the CNN model is capturing the main pattern of the data.
- There are areas, such as around 06:52:00 and again near 11:52:00, where the predicted values show significant volatility compared to the ground truth. This might indicate the model is somewhat overfit to certain features in the data or is responding too strongly to noise.
- The CNN seems to detect and react to the major spikes in outbound utilization, particularly the one near 10:52:00. However, the model does not capture the full intensity of the spikes, underestimating the peak utilization.
- Compared to traditional time series models (like ARIMA), the CNN appears to produce a smoother prediction line, which might be due to its ability to capture more complex patterns in the data through its layered structure.
- Despite the volatility, the CNN model's predictions are close to the actual values for most of the time series.
- The model seems to exhibit minor lagging at certain points, for example, following the 07:52:00 mark. However, there are no significant phases where the predicted values consistently lag or lead the actual values by a large margin.

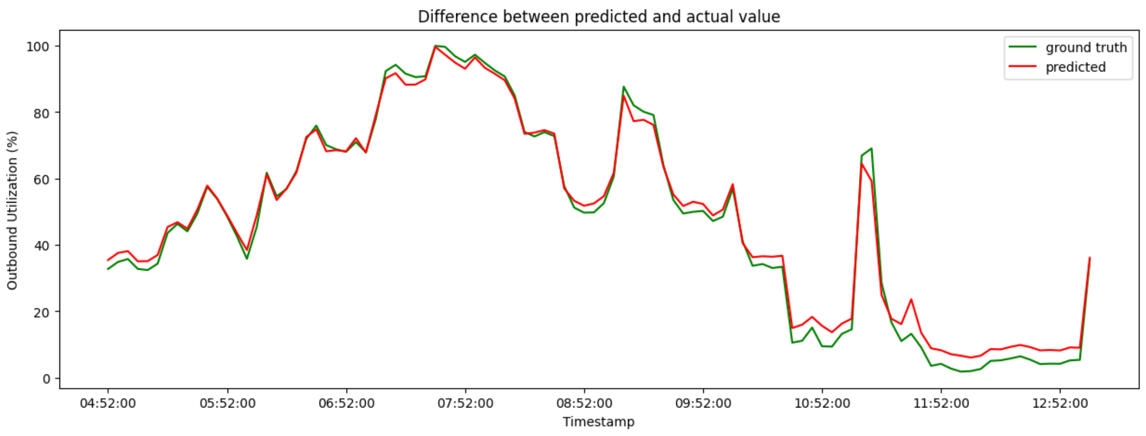

Figure 5.7: Difference between actual and predicted value of the CNN model.

5.2.2 Gated recurrent unit (GRU) network

The architecture depicted in Figure 5.8 consists of a sequence of layers organized to process an input tensor and produce an output tensor.

- Input Tensor: The input to the network has a shape of (1, 1, 2), where the dimensions represent a batch size of 1, a feature size of 1, and a sequence length of 2, based on Figure 5.9.

- GRU Layer: This layer processes the input tensor. The GRU expands the input shape to (1, 1, 59) and outputs two tensors with shapes (1, 1, 59) and (3, 1, 59), indicating 59 hidden units, based on Figure 5.10, and three layers within the GRU network, based on Figure 5.11.
- *getitem* Operation: This operation selects a specific output from the GRU layer, reducing the tensor shape from (1, 1, 59) to (1, 59), which is then fed into the following layer.
- Linear Layer: This layer transforms the 59-dimensional input into a single scalar value, acting as the final stage of computation.
- Output Tensor: The final output tensor of the network has a shape of (1, 1), which is the result of processing the input tensor through the GRU network.

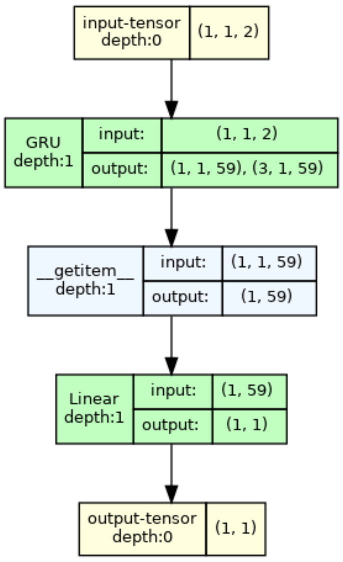

Figure 5.8: Architecture of the GRU network implementation.

Hyperparameters

The Gated Recurrent Unit (GRU) network, a variant of recurrent neural networks, is governed by several key hyperparameters that significantly impact its performance. Among these, the input sequence length is the most significant as it determines the number of timesteps the GRU considers for each input, thus affecting how much of the temporal context the model can capture. The number of recurrent layers in the network defines the depth of the model. More layers can enable the GRU to learn complex patterns but may also lead to increased computational complexity and risk of overfitting. Lastly, the number of features in the hidden state, often referred to as the size of the hidden layer, is crucial because it represents the capacity of the network to store information across timesteps. This dimensionality of the hidden state directly influences the ability of the GRU to remember and utilize past information, which is essential for tasks involving sequential data.

Figure 5.9 shows how the size of the input sequence affects the RMSE. The line representing RMSE shows a highly variable pattern as the input length increases. There is a sharp decline from the first data point, followed by an immediate sharp increase. Throughout the graph, there are several peaks and troughs indicating fluctuation in RMSE with different input lengths. The overall trend does not suggest a clear linear relationship. Instead, it shows that the performance of the GRU model in terms of RMSE varies significantly with the change in input length. At certain sequence lengths, the GRU model can get better at learning from data dependencies. This better learning power at specific sequence lengths can result in a more accurate model, which has a lower RMSE.

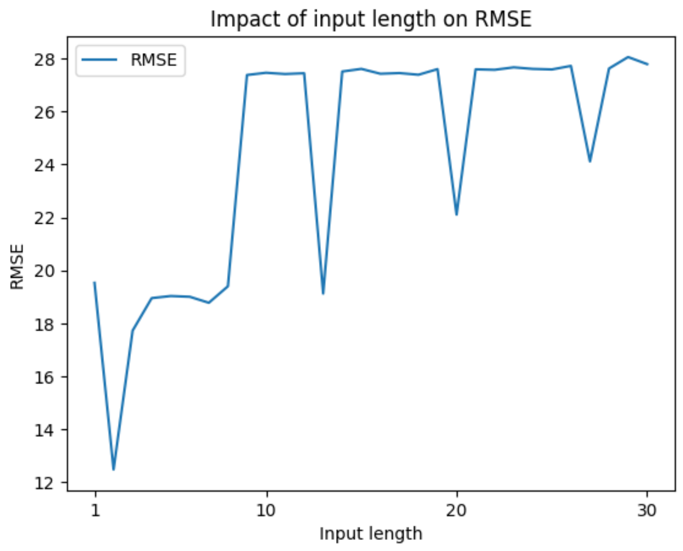

Figure 5.9: Dependence between GRU input sequence size and RMSE.

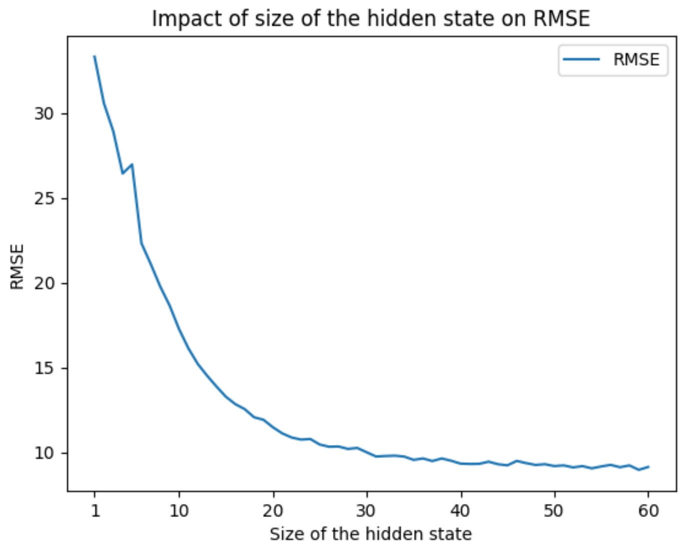

Figure 5.10: Dependence between GRU size of the hidden state and RMSE.

Graph in the Figure 5.10 shows a sharp decrease in RMSE as the size of the hidden state increases from 1 to around 20. Beyond this point, the decrease in RMSE slows down significantly, eventually stabilizing as the hidden state size approaches 60. This indicates that larger hidden state sizes can improve model accuracy, although the returns diminish after a certain threshold. The flattening of the curve suggests that increasing the size beyond 50 provides minimal additional benefits in terms of reducing RMSE.

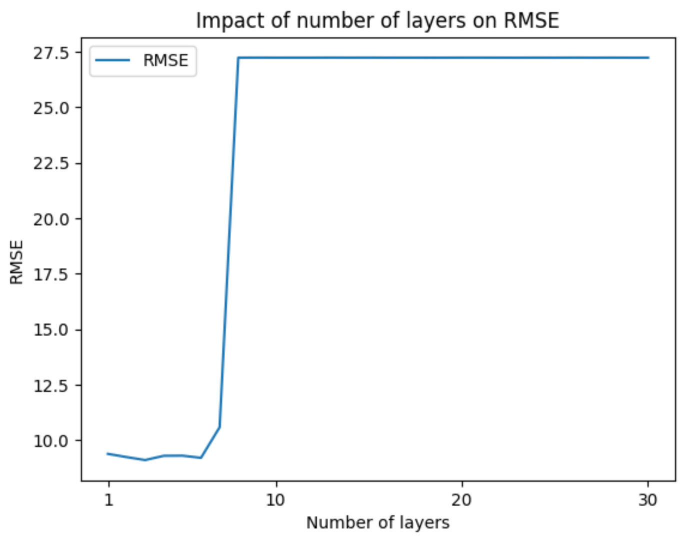

Figure 5.11: Dependence between GRU number of hidden units and RMSE.

Figure 5.11 presents a dependence between the number of GRU cell and RMSE. A significant drop in RMSE is observed when the number of layers increases from 1 to just under 10. After this initial decrease, the RMSE stabilizes, showing minimal variation as the number of layers continues to increase up to 30. This stabilization suggests that adding more layers beyond a certain threshold does not significantly enhance predictive accuracy of the model.

Input length	2
Hidden size	59
Number of layers	3
Batch size	64
Learning rate	0.001
Training epochs	100

Table 5.2: Hyperparameters of the GRU model.

Table 5.2 presents a set of hyperparameters of the Gated Recurrent Unit (GRU) neural network model. These hyperparameters are:

- Input length: This parameter indicates that the GRU model takes sequences of length 2 as input for each prediction.
- Hidden size: The hidden size refers to the number of units in each GRU layer. A size of 59 means that internal representation of the data at each LSTM layer will have 59 dimensions.
- Number of layers: The model comprises three GRU layers.

- Batch size: This denotes that 64 samples from the dataset are used to estimate the error gradient before the weights of the model are updated during training.
- Learning rate: The rate at which the model learns during training.
- Training epochs: The model is set to run through the entire training dataset 100 times during the training process.

Forecasting

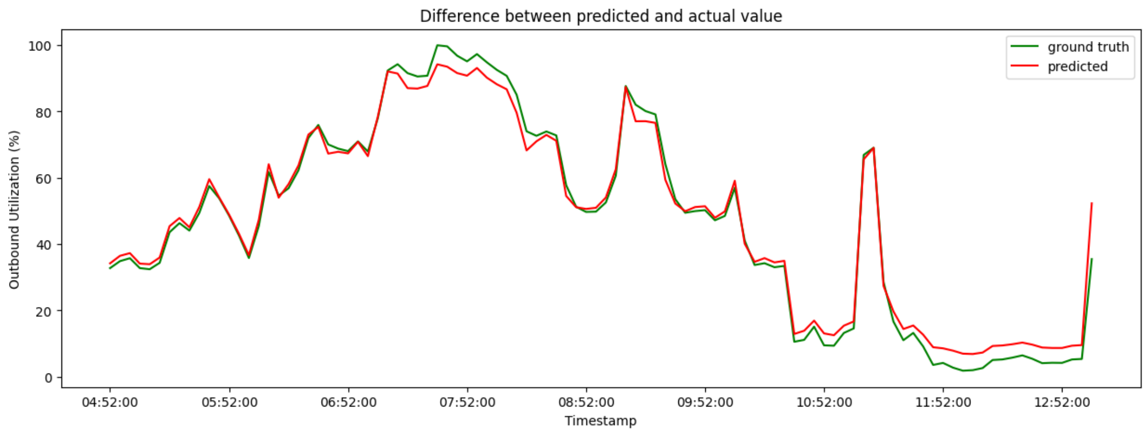

Figure 5.12: Difference between actual and predicted value of the GRU model.

Figure 5.12 shows the performance of a GRU network compared to the actual values.

- The GRU network seems to track the general trends of the actual data quite well, as the overall shapes of the predicted and ground truth lines are similar. This indicates that the network has learned the underlying pattern.
- The major deviations between the predicted and actual values occur around 06:52 and after 11:52, which could be due to unexpected changes that the model did not anticipate.

5.2.3 Long Short-Term Memory (LSTM) network

The LSTM architecture in the Figure 5.13 consists of the following layers and operations:

- Input Tensor: The network starts with an input tensor of shape (1, 1, 2), where the dimensions represent a batch size of 1, a feature size of 1, and a sequence length of 2, as this value yields the lowest error, shown in Figure 5.14.
- LSTM Layer: The input tensor is then passed to an LSTM layer. This layer transforms the input to an output with the shape of 3 x (1, 1, 5). This indicates that three separate tensors are generated for each time step, representing the hidden state, the cell state, and the output. 5 as the value of hidden state is based on the Figure 5.16.
- *getitem* Operation: The network performs a getitem operation to extract one tensor from the LSTM output, specifically selecting the LSTM cell output.

- Linear Layer: The extracted tensor is then fed into a Linear layer, which applies a linear transformation to the data, resulting in a tensor of shape (1, 1). This step is designed to map the features from a higher-dimensional space to a single value.

- Output Tensor: The final output of the network is a tensor of shape (1, 1), representing the prediction.

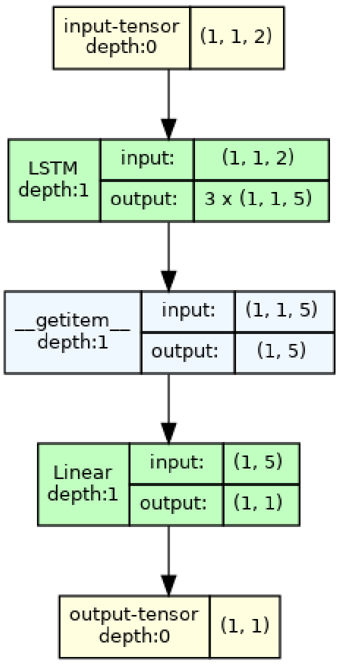

Figure 5.13: Architecture of the LSTM network implementation.

Hyperparameters

Three key hyperparameters stand out while configuring an Long Short-Term Memory (LSTM) network: input sequence length, the number of hidden units and the number of features in the hidden state. The input sequence length, which determines the number of time steps the LSTM processes before making a prediction, directly impacts the amount of temporal information that can be utilized by the model and by adjusting this parameter, model can be helped to capture more or less temporal context. On the other hand, the number of hidden units in each LSTM cell influences the capacity of the model to learn and represent complex patterns. An increase in hidden units can provide a richer representation but may also raise the risk of overfitting and computational costs. Lastly, the number of features in the hidden state determines the dimensionality of the output and state vectors of each LSTM cell, affecting the granularity at which the network processes and remembers information.

Figure 5.14 exhibits a trend where the RMSE sharply fluctuates at the lower end of the input length spectrum, showing a significant drop (indicating better performance) before it climbs back up. The RMSE peaks at around an input length of 10 before it begins to fluctuate. From an input length of approximately 15 and beyond, the RMSE seems to stabilize, showing minor fluctuations but maintaining a consistent value. This suggests that beyond a certain input length, the improvement in error reduction is marginal or non-existent. The sharp peak in RMSE around an input length of 3 suggests inadequate contextual information at this sequence size, leading to higher prediction errors. This could be due to model sensitivity where short sequences do not capture enough temporal dependencies.

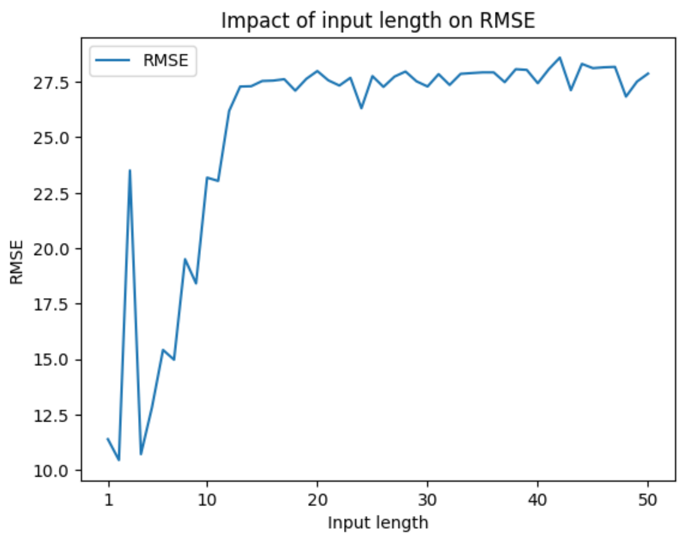

Figure 5.14: Dependence between LSTM input sequence size and RMSE.

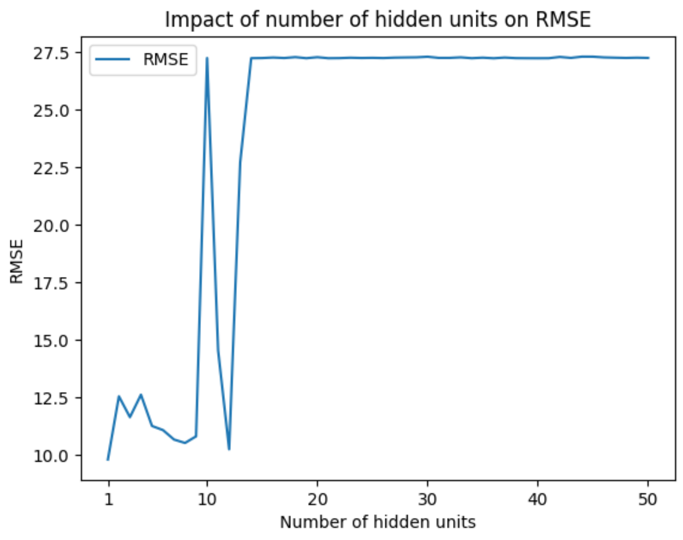

Figure 5.15: Dependence between LSTM number of cells and RMSE.

Figure 5.15 illustrates an initial variability in RMSE as the number of hidden units increases. The RMSE peaks sharply at approximately 5 hidden units before quickly declines, indicating enhancement in performance. Following this, the RMSE slightly rises and then levels off. From 10 hidden units onward, the RMSE peaks and remains fairly constant, with minimal fluctuations. This behavior indicates that beyond a certain number of hidden units, additional increases do not significantly affect RMSE, suggesting the existence of an optimal number of hidden units above which no substantial improvements in error reduction are observed. The significant decrease in RMSE at around 11 hidden units highlights a critical improvement in model accuracy, likely indicating that the model has achieved a sufficient number of hidden units to adequately capture the data complexity. Having too few hidden units could result in underfitting, where the model is unable to effectively learn the underlying patterns of the data. On the other hand, further increasing the hidden units past this point does not seem to markedly influence the RMSE, implying that capacity of the model has stabilized and additional complexity does not lead to additional gains.

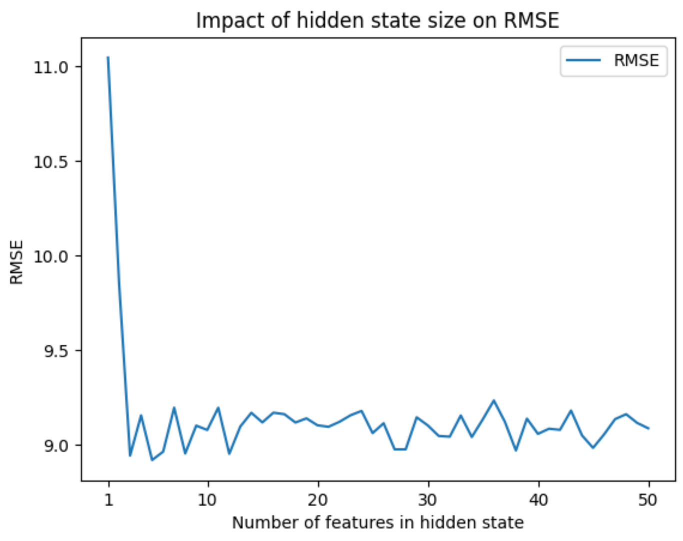

Figure 5.16: Dependence between LSTM size of the hidden state and RMSE.

Figure 5.16 illustrates the relationship between the size of the hidden state in an LSTM network and the RMSE. Starting from 1 feature, there is a steep decrease in RMSE as more features are added. After this initial drop, the RMSE begins to fluctuate, indicating smaller improvements in error reduction as more features are added to the hidden state. The overall trend suggests that increasing the hidden state size improves the accuracy up to a certain point, after which the benefits diminish.

Input length	2
Hidden size	5
Number of layers	1
Batch size	64
Learning rate	0.001
Training epochs	100

Table 5.3: Hyperparameters of the LSTM model.

The table 5.3 shows the hyperparameters chosen for training the Long Short-Term Memory (LSTM) neural network model:

- Input length: This specifies that the LSTM network takes input sequences with a length of 2. Each sequence input to the model contains 2 time steps of data.

- Hidden size: This number refers to the dimensionality of the hidden state within the LSTM cells. With a size of 5, each LSTM cell will output a hidden state vector with 5 elements.
- Number of layers: The LSTM model consists of a single layer.
- Batch size: The model will process 64 training examples at a time before updating the weights. This batch size is a compromise between computational efficiency and the stability of the learning process.
- Learning rate: The rate at which the model learns during training.
- Training epochs: This indicates that the entire dataset will be passed through the LSTM network 100 times in total during training.

Forecasting

The figure 5.7 presents a comparison between actual data points and predicted values of the LSTM model.

- The predicted values follow the ground truth closely for most parts, which suggests that the LSTM model has learned the pattern of the time series reasonably well.
- There are points, specifically around 07:52:00, where the predictions diverge significantly from the actual data. These spikes could indicate sudden changes that the model was unable to predict.
- Towards the end of the time span, the prediction starts to deviate more from the actual measurements, with the model overestimating before 12:52:00.

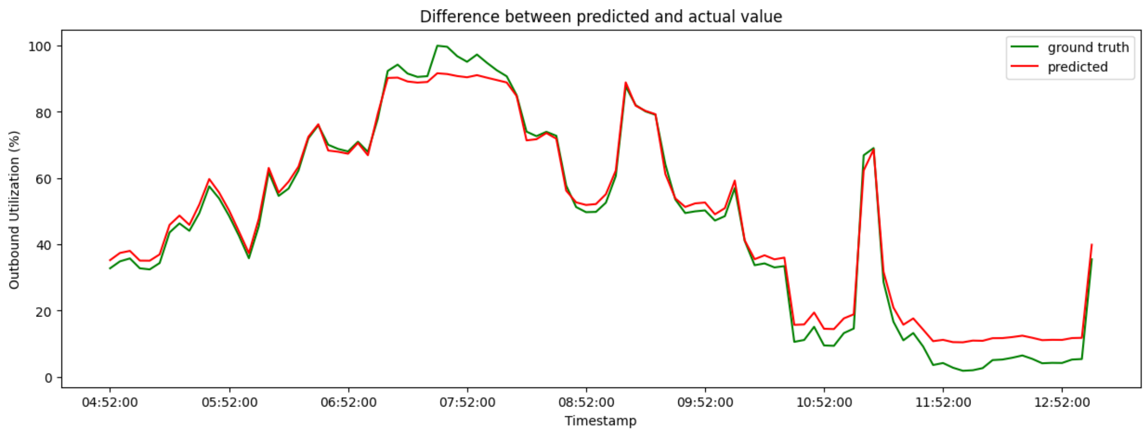

Figure 5.17: Difference between actual and predicted value of the LSTM model.

5.3 Results

Model	MAE	MAPE (%)	SMAPE (%)	RMSE
MA	2.39	48.37	7.76	3.01
AR	27.37	70.68	54.26	33.96
ARIMA	14.77	0.37	29.93	18.15
CNN	3.73	0.33	24.46	4.17
GRU	3.24	0.40	29.17	3.79
LSTM	2.25	28.49	8.49	2.25

Table 5.4: Metrics results

The Table 5.4 presents performance metrics of the selected models.

Within the scope of Mean Absolute Error (MAE), it is indicated that the LSTM model has achieved the lowest value, signifying minimal average errors in its predictions, while the AR model forecasts are associated with a considerably higher average error, as reflected by the highest MAE value. The MA and GRU models also demonstrate relatively low MAE values, suggesting better accuracy than the AR and ARIMA models but not as accurate as the LSTM model.

In terms of Mean Absolute Percentage Error (MAPE), which evaluates forecast accuracy as a percentage, a remarkably low MAPE for the ARIMA model implies exceptional accuracy, and the CNN model also shows a commendably low MAPE. In stark contrast, the AR model forecasts have a significantly higher percentage error, with the MAPE being the largest among the models, and the MA model also shows a relatively high MAPE despite its low MAE.

The Symmetric Mean Absolute Percentage Error (SMAPE) addresses the shortcomings of MAPE by accounting for the asymmetry in the percentage errors. The CNN model achieves the lowest SMAPE, indicating a balanced distribution of forecast errors, whereas the AR model registers the highest SMAPE, suggesting its errors are notably asymmetric. The GRU and LSTM models also perform well on this metric, with SMAPE values indicating relatively symmetric and small percentage errors.

Lastly, the Root Mean Square Error (RMSE) penalizes larger forecast errors by squaring the error values prior to averaging. The LSTM model performance is exemplary with the lowest RMSE, denoting a small dispersion of errors, particularly larger ones. High RMSE of the AR model signifies a greater dispersion of substantial errors. The MA, CNN, and GRU models present moderate RMSE values, which implies they are generally reliable but may occasionally produce significant errors.

Overall, the LSTM model is consistently highlighted as the most accurate across all metrics, suggesting robust performance in time series forecasting. The AR model, with the highest values in all metrics, is indicated to have the largest forecast errors, questioning its suitability for the dataset. The ARIMA model, despite its stellar performance in MAPE, does not exhibit uniformity across the metrics. The CNN and GRU models offer a balance, with good performance in SMAPE and RMSE, indicating they might be reliable alternatives, particularly in contexts where percentage errors and error symmetry are of concern. The MA model performance suggests that while it may have a low average error, the percentage errors can be relatively high, necessitating careful consideration of the specific context in which it is applied.

5.4 Summary

In this chapter, traditional statistical models such as Moving Average (MA), Autoregressive (AR), and Autoregressive Integrated Moving Average (ARIMA) were applied to the data, with detailed discussions on model selection and parameter tuning. This was followed by the implementation of advanced machine learning techniques, including Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) such as GRU and LSTM, which were explored for their ability to capture complex data patterns. The effectiveness of these models was evaluated using performance metrics, with comparative analyses presented to highlight capabilities of each model in network traffic forecasting.

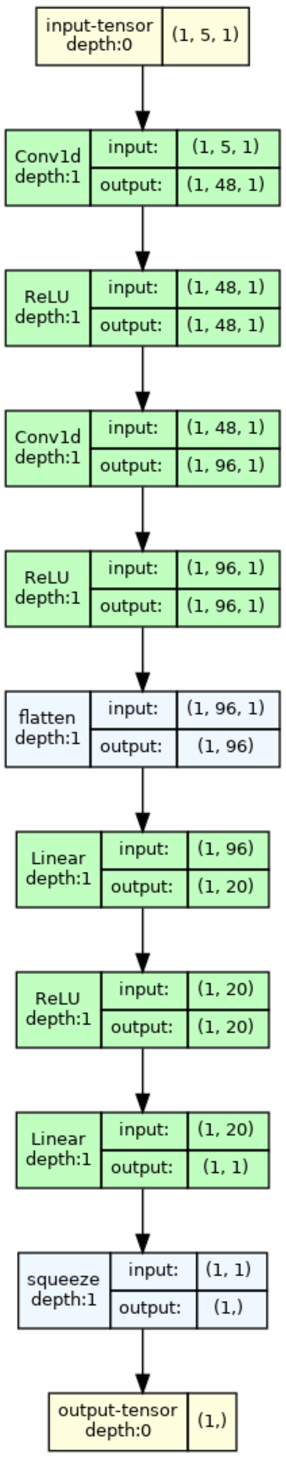

Figure 5.5: Architecture of the CNN implementation.

Chapter 6

Usability and efficiency

The shift of machine learning models from experimental to production contexts in network communication necessitates not only strong accuracy, but also great operational efficiency and flexibility. This chapter covers the applicability of machine learning models in production contexts and investigates several ways for speeding up the training of these models, making them more suitable for real-time applications.

6.1 Usability in production

Implementing machine learning models in a live network context also poses practical issues, such as dealing with real-time data streams, guaranteeing data privacy and security, and controlling resource allocation. Solutions like putting models in virtualized settings or utilizing cloud-based services can assist to reduce these issues by offering scalable and secure model operating resources.

6.1.1 Integration

Effective integration involves ensuring that the model communicates seamlessly with the current network infrastructure. This often necessitates the creation of special APIs that facilitate data interchange between the model and network monitoring tools. Additionally, scalability of the model must be considered in order to accommodate variable amounts of network traffic data. The seamless integration of machine learning models into existing network systems is critical.

6.1.2 Performance monitoring and maintenance

Once deployed, the model performance must be periodically monitored to verify that it satisfies the required accuracy and efficiency standards. This involves setting up automated performance tracking to detect any loss in prediction accuracy that may be induced by changes in network traffic patterns. Regular model updates and retraining with fresh data are essential for adapting to such changes and maintaining peak performance.

6.2 Accelerating model training

Accelerating the training of machine learning models is critical for rapid deployment and model updates in production.

Model	Training time	Device
Moving Average (MA) model	5 min 37 sec	CPU
Autoregressive (AR) model	0 sec	CPU
ARIMA model	10 min	CPU
Convolutional Neural Network (CNN)	9 min 26 sec	GPU
Gated recurrent unit (GRU) network	50 sec	GPU
Long Short-Term Memory (LSTM) network	4 min 28 sec	GPU

Table 6.1: Training time and device used to train model.

Models in Table 6.1 were trained on Ryzen 9 7900X3D and Nvidia RTX 4080 Super. In production, the choice of model would depend on various factors such as the required accuracy, the time sensitivity of the predictions, resource availability, and the specific characteristics of the time series data. Some of the methods for reducing the training time include:

- Hardware Improvements: Training on more powerful GPU, using distributed computing, or specialized hardware like TPU (Tensor Processing Unit) could reduce training time.
- Model Complexity: Reducing the number of parameters, layers, or the input size could lead to faster training, though at the potential cost of accuracy.
- Batch Size: Increasing the batch size can lead to faster training as more data is processed in parallel, but this may also require more memory and can affect model performance.
- Learning Rate and Optimizers: Adjusting the learning rate or using different optimization algorithms can affect the convergence speed of the training process.

6.3 Summary

This chapter addressed the usability and efficiency of machine learning models in network communications production scenarios. It discussed how to integrate these models into current infrastructure, addressing issues such as data stream handling, privacy, security, and resource management. The chapter highlighted the importance of performance monitoring and maintenance, arguing for automated performance tracking systems as well as the need for regular model updates and retraining with fresh data to ensure accuracy. It also investigated ways for accelerating model training, including hardware upgrades, architecture optimization, and hyperparameter tweaking, with the goal of enabling rapid deployment and ensuring models could adapt to changing network conditions.

Chapter 7

Conclusion

This thesis has explored the application of machine learning techniques to forecasting of time series data in the context of network communications, a critical area for maintaining and optimizing modern network systems. By integrating both traditional statistical models, such as ARIMA, and advanced machine learning approaches, including Long Short-Term Memory (LSTM) and Convolutional Neural Networks (CNNs), it has demonstrated the potential for significant improvements in predictive accuracy and efficiency.

The thesis findings show that, while classic approaches like ARIMA are effective at addressing seasonal data patterns, they frequently fall short when dealing with complex, non-seasonal data relationships found in network traffic. In contrast, LSTM and CNN models have demonstrated greater capacity to catch these complex patterns, due to their deep learning capabilities and sophisticated structures to handle and learn from massive volumes of data.

This study shows that machine learning models, particularly LSTM, are better suited to managing the dynamic and unpredictable character of network data, offering a viable alternative to traditional forecasting approaches. These models not only improve forecast accuracy, but they also help to improve network management efficiency by allowing for proactive reactions to predicted traffic situations.

However, the transition to machine learning based forecasting is not without its problems. The intricacy of these models necessitates significant computational resources as well as experience in hyperparameter selection and interpretation. Furthermore, the black-box aspect of many machine learning models might make them less appealing to practitioners who prefer transparent, interpretable models.

Future research should include hybrid models that combine the interpretability and simplicity of classic statistical approaches with the predictive capability of machine learning techniques. Furthermore, future research might focus on improving the scalability and efficiency of these models, making them more useful for real-time network management applications. As network settings change, forecasting model flexibility and adaptability will become increasingly important in fulfilling the growing needs for network performance and security.

Bibliography

[1] *Augmented Dickey-Fuller* [online]. 2023 [cit. 2023-12-18]. Available at: https://www.statsmodels.org/dev/generated/statsmodels.tsa.stattools.adfuller.html.

[2] Alto, V. *Understanding Ordinary Least Squares (OLS) Regression* [online]. 2023 [cit. 2024-03-08]. Available at: https://builtin.com/data-science/ols-regression.

[3] Brockwell, P. J. and Davis, R. A. *Introduction to Time Series and Forecasting*. 2nd ed. Springer, 2002. ISBN 0-387-95351-5.

[4] Burnham, K. and Anderson, D. *Model Selection and Multimodel Inference: A Practical Information-Theoretic Approach*. Springer New York, 2003. ISBN 9780387953649.

[5] Chunduri. *Understanding LSTM Internal blocks and Intuition* [online]. 2020 [cit. 2024-03-12]. Available at: https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[6] Chung, J., Gülçehre Çaglar, Cho, K. and Bengio, Y. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. *ArXiv*. 2014, abs/1412.3555. Available at: http://arxiv.org/abs/1412.3555.

[7] Cokelaer, T., Kravchenko, A., Varma, A., L, B., Stringari, C. E. et al. *Cokelaer/fitter: v1.7.0*. Zenodo, january 2024. DOI: 10.5281/zenodo.10459943. Available at: https://doi.org/10.5281/zenodo.10459943.

[8] Goodfellow, I. J., Bengio, Y. and Courville, A. *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.

[9] Hamilton, J. D. *Time Series Analysis*. United States: Princeton University Press, 1994.

[10] Hochreiter, S. and Schmidhuber, J. Long Short-Term Memory. *Neural Computation*. November 1997, vol. 9, no. 8, p. 1735–1780. ISSN 0899-7667.

[11] Hyndman, R. J. Moving averages. November 2009, [cit. 2023-11-01]. Available at: https://robjhyndman.com/papers/movingaverage.pdf.

[12] Hyndman, R. and Athanasopoulos, G. *Forecasting: Principles and Practice*. 2nd ed. Australia: OTexts, 2018.

[13] Johnson, N., Kotz, S. and Balakrishnan, N. *Continuous Univariate Distributions, Volume 2*. New York: Wiley, 1995. Wiley Series in Probability and Statistics. ISBN 9780471584940.

[14] Kang, C. *Autoregressive (AR) models.* Jun 2020. Available at: https://goodboychan.github.io/python/datacamp/time_series_analysis/2020/06/08/01-Autoregressive-Models.html.

[15] Kiran, T. *Unlocking the Potential of Convolutional Neural Networks (CNNs) in Time Series Forecasting* [online]. 2023 [cit. 2024-03-08]. Available at: https://bookdown.org/rdpeng/timeseriesbook/.

[16] Kotzé, K. *State-space modelling.* Available at: https://kevinkotze.github.io/ts-4-state-space/.

[17] Massey, F. J. The Kolmogorov-Smirnov test for goodness of fit. *Journal of the American Statistical Association.* United States: American Statistical Association. 1951, vol. 46, no. 253, p. 68–78.

[18] Monigatti, L. *Interpreting ACF and PACF Plots for Time Series Forecasting* [online]. 2022 [cit. 2023-12-21]. Available at: https://towardsdatascience.com/interpreting-acf-and-pacf-plots-for-time-series-forecasting-af0d6db4061c.

[19] Nielsen, A. *Practical Time Series Analysis.* 1st ed. Sebastopol: O'Reilly Media, Inc., 2019.

[20] Olah, C. *Understanding LSTM Networks* [online]. 2015 [cit. 2024-03-09]. Available at: https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[21] Pattnaik, S. *Network Analytics Time Series* [online]. 2020 [cit. 2023-11-07]. Available at: https://www.kaggle.com/datasets/pattnaiksatyajit/network-analytics-time-series/data.

[22] Peng, R. D. *A Very Short Course on Time Series Analysis* [online]. 2020 [cit. 2024-03-08]. Available at: https://bookdown.org/rdpeng/timeseriesbook/.

[23] Sampaio, V. *Understanding ordinary least squares (OLS): The foundation of linear regression.* Medium, Jun 2023. Available at: https://medium.com/@VitorCSampaio/understanding-ordinary-least-squares-ols-the-foundation-of-linear-regression-1d79bfc3ca35.

[24] Sarra, D. *How to interpret SMAPE just like MAPE* [online]. 2022 [cit. 2023-12-21]. Available at: https://medium.com/@davide.sarra/how-to-interpret-smape-just-like-mape-bf799ba03bdc.

[25] Thakur, A. *Intuitive understanding of 1D, 2D, and 3D convolutions in convolutional neural networks.* [online]. [cit. 2024-03-08]. Available at: https://wandb.ai/ayush-thakur/dl-question-bank/reports/Intuitive-understanding-of-1D-2D-and-3D-convolutions-in-convolutional-neural-networks---VmlldzoxOTk2MDA.

[26] Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T. et al. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods.* 2020, vol. 17, p. 261–272. DOI: 10.1038/s41592-019-0686-2.

[27] Zheng, A. *Evaluating Machine Learning Models: A Beginner's Guide to Key Concepts and Pitfalls.* O'Reilly Media, 2015. ISBN 9781491932469.

Appendix A

Contents of the included storage media

- **data/** - folder containing the dataset
- **utils/** - folder containing utilities to determine hyperparameters
- **models/** - folder with trained models
- **thesis/** - folder containing the LaTeX source codes and pdf containing text of the thesis
- **outputs/** - outputs of the hyperparameters search
- **Network_Analytics.ipynb** - implementation of the methods described in this thesis
- **README.md** - instructions for setting up the environment
- **requirements.txt** - dependencies on the third party libraries