FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

MASTER THESIS

Samuel Bartoš

Prediction of energy load profiles

Department of Theoretical Computer Science and Mathematical Logic

Supervisor of the master thesis:	RNDr. Jiří Fink, Ph.D.
Study programme:	Computer Science
Study branch:	Artificial Intelligence

Prague 2017

Title: Prediction of energy load profiles

Author: Samuel Bartoš

Department: Department of Theoretical Computer Science and Mathematical Logic

Supervisor: RNDr. Jiří Fink, Ph.D., Department of Theoretical Computer Science and Mathematical Logic

Abstract: Prediction of energy load profiles is an important topic in Smart Grid technologies. Accurate forecasts can lead to reduced costs and decreased dependency on commercial power suppliers by adapting to prices on energy market, efficient utilisation of solar and wind energy and sophisticated load scheduling. This thesis compares various statistical and machine learning models and their ability to forecast load profile for an entire day divided into 48 half-hour intervals. Additionally, we examine various preprocessing methods and their influence on the accuracy of the models. We also compare a variety of imputation methods that are designed to reconstruct missing observation commonly present in energy consumption data.

Keywords: time series, state-space models, neural networks, imputation, preprocessing

None of this would be possible without the guidance and patience of my thesis supervisor RNDr. Jiří Fink, Ph.D. I would also like to thank my family for their support. Finally, I want to express my gratitude to those who went above an beyond to help me, including Juraj Citorík, Zuzana Straková and Martin Novák.

Contents

1. Introduction

Our current electric grid was conceived more than a hundred years ago [1] as a collection of centralised unidirectional systems designed to meet very simple energy demands. However, the modern household is very different from its hundred-year-old counterpart. Its energy demands grew from a couple of light bulbs and a radio to include a large number and variety of appliances. Moreover, new renewable energy sources such as wind turbines and solar panels are unpredictable when it comes to their energy output profiles. This has rendered the grid inefficient if not obsolete.

Smart Grid [2] is a set of technologies designed to eliminate or at least alleviate these shortcomings. A common element to most definitions of Smart Grid is the application of computer processing, automation and two-way communication technology to the power grid, making it possible to adjust and control devices individually from a central location.

One of the problems Smart Grid faces is forecasting energy consumption of a household [3]. Reliable forecasts can be used by specialised algorithms to anticipate peak consumptions and prevent penalties imposed by electricity companies on exceeding the limit previously agreed in energy supply contract. These can also help a village with its own energy sources to become less energy dependant on outside suppliers by synchronizing individual households consumption profiles.

A common approach to forecasting energy consumption is time series analysis. Time series is in essence a sequence of observations whose value varies through time. Time series analysis tries to uncover or at least approximate the hidden process that generated these observations. When the true nature of the underlying process is exposed, it can be used to forecast its future behaviour [4].

Apart from load forecasting, there are many other problems in Smart Grid suitable for time series analysis. For example, tracking the fluctuations of the price of energy on the energy market can help a sophisticated system to adapt by prepurchasing and storing temporarily cheap energy for later use or delaying energetically expensive operations while the current price is high [5]. Also the more the use of solar power as a source of electricity in Smart Grids increases, the more important the forecasts of solar irradiance becomes. For instance, managing and operating a solar power plants with energy storage system requires such reliable forecast [6]. The same is true for wind-based power plants [7].

One of the most important factors to consider when analysing time series is the quality and quantity of data provided for the training purposes. First of all, energy demands are considered a private information, which results in them not being readily available and thus lowering the quantity. Secondly, the quality of data goes hand in hand with the purpose of the current electric grid. Because the grid is currently only set up to transfer energy to its customers and bill them on a monthly basis, there is no incentive for companies responsible for energy production to provide anything more than total monthly loads aggregated across all devices and appliances. Data of such a low granularity is of very limited use when it comes to time sereis analysis. However, recent developments have introduced Smart Meter, a device able to record and communicate energy consumption in intervals of an hour or less to the central system [8]. This thesis seeks to, among others, utilise this feature.

The performance of various time series analysis techniques can be improved by suitable preparation of the data in a process called preprocessing [9]. The preprocessing is done before the analysis in order to simplify the patterns present in the data by making them more consistent across the whole data set or by removing known sources of variation. This leads to better analysis because simpler patters are more easily reproduced in mathematical models. The effectiveness of a preprocessing method may depend on the particularities and aims of analysis and also on the characteristics of the data. One of the goals of this thesis is to examine various preprocessing methods and their effectiveness as it relates to forecasting energy load profiles.

No electromechanical device is perfect and Smart Meters are no exception. The failure of this device results in missing observations. Another source of missing observations are electrical outages that happen from time to time in any electrified dwelling. The problems the missing observations introduce include technical difficulties when creating and estimating various models, addition of a substantial amount of bias to the analysis and reductions of the models' accuracy. For these reasons it is a common practice to estimate the values of missing observations from the values of other observations in a process called imputation. After missing observations are imputed, the analysis continues using standard techniques for complete data. The more accurate the imputations are with respect to real unobserved values, the better the analysis. Therefore, there is a need for examining various imputation methods in terms of their respective accuracies.

1.1 Goals

We summarise the primary goals of this thesis in the following list:

- describe the theoretical background of various techniques used in time series analysis
- compare various imputation methods,
- examine the accuracy of a variety of forecasting models with respect to load forecasting,
- study the impact of preprocessing methods on the models' accuracy.

1.2 Methods

There is a number of techniques frequently used in time series analysis. This thesis focuses on a selection of statistical methods and machine learning models, and compares their advantages, disadvantages and performance.

The statistical methods considered in this thesis are all based around state-space models. State-space models [10] are a representation of physical systems. They regard observations in time series as measurements of the hidden state of the system (signal) corrupted by noise. The current state of the system is not measured directly, but it is estimated from past noisy observations. State-space models contain a number of parameters that must be estimated before making predictions. One advantage of state-space models and the statistical theory that supports them is that this estimation can be done in an objective manner using methods of statistical inference [4]. Another advantage of state-space models is that their white-box nature. This means that one can gain an understanding of the inner-workings of the hidden process by examining the equations and hidden-state representation of a state-space model.

State-space models differ from one another by their representation of the hidden state of the system. One type of state-space models considered in this thesis are exponential smoothing (ES) models [11]. ES models decomposes the process or time series into level, trend and seasonality components. These components are not modelled separately, but are interconnected and affect each other.

On the other hand, another type of state-space models, called autoregressive moving average (ARMA), tries to preprocess the time series in such a way as to remove these components entirely. What is left is treated as a combination of two regression models. The first one regresses current observation against past observations. The second one is less intuitive. It regresses current observation against past noise. The motivation stems from the assumption that whatever corrupted the observations of system's state by a large noise in recent past will continue to affect the observations of system's state in the near future. For example, an unexpected electrical outage may cause big discrepancies between expected load consumption derived from the state of the system and actual observed load consumption. This outage is believed to cause similar discrepancies in the near future.

ES and ARMA models use past observation as the only source of information for the model of the system. However, time series in general and energy consumption in particular are usually affected also by other sources. In our case the most pertinent source of information is weather data [12]. In contrast, regression models in general aim to expose the relationship between various sources of information and explain their influence on the dependant variable but may fail to capture the subtle dynamics of time series targeted by ES or ARMA models. For this reason we also consider a combination of weather regression with ARMA model called ARMA with exogenous inputs (ARMAX) in order to take advantage of both approaches.

Machine learning [13] focuses on algorithms that are able to enhance their performance by learning from past experiences. In time series analysis, a machine learning method is iteratively presented with sample inputs (a number of past data point) and the desired output (current data-point). The aim is to teach the algorithm the general rule that maps the inputs to outputs. This thesis explores, among others, the effectiveness of a type of machine learning algorithm called neural networks.

Neural networks has been used to successfully solve multitude of engineering problems, from predictions of heart attacks [14] and stock market prices [15] to credid card fraud detection [16] and self-driving cars [17]. Their strength lies mainly in their ability to model hidden nonlinear patterns that are too complex to be detected by humans or other computer methods. Through an iterative process of learning, neural network is taught various characteristics of a target system. This also means that it is able to adapt automatically, making it in theory viable for predicting ever-changing household energy demands. However, neural network are at a disadvantage when it comes to uncovering the inner-workings of the process generating the time series undergoing analysis, because of their inherent black-box nature.

1.3 Structure

The structure of this thesis is following. Chapter 2 contains notation and definitions used throughout this thesis. Chapter 3 describes load forecasting and analyses the energy consumption data. In Chapter 2 we review published work related to forecasting energy consumption. Then in Chapter 5 we present the theory behind state-space models. This is followed by three chapters describing different state-space models: Chapter 6 is dedicated to ES models, Chapter 7 focuses on ARMA models and Chapter 8 is devoted to ARMAX models. In contrast to state-space models, the machine learning approach to time series analysis using neural networks is described in Chapter 9. Chapter 10 focuses on methodology used when conducting experiments whose results are presented in Chapter 11 for imputation and Chapter 12 for forecasting. The conclusions from the results are drawn in Chapter 13.

2. Preliminaries

This chapter introduces the problem of time series analysis, notation and definitions used throughout this thesis.

2.1 Notation

Throughout this thesis we use regular font for scalar values as in x_t and bold font for vectors as in $\boldsymbol{x}$ or $\boldsymbol{\theta}$. In particular, let $\mathbf{0}_k$ be a vector of k zeros. The bold fond symbols and regular symbols are always related, meaning that x_t is always an element of $\boldsymbol{x}$ without specifically mentioning this fact. For simplicity, unless stated otherwise, we consider all vectors to be column vectors, even when writing $\boldsymbol{x} = (x_1, x_2, \ldots, x_k)$. When we want to specify a row vector we use transposition operator denoted by $'$ as in $\boldsymbol{x}'$. Furthermore, let $\odot$ denote the *Hadamard product* (element-wise multiplication) of vectors $\boldsymbol{x} = (x_1, x_2, \ldots, x_k)$ and $\boldsymbol{y} = (y_1, y_2, \ldots, y_k)$, i.e. $\boldsymbol{x} \odot \boldsymbol{y} = (x_1y_1, x_2y_2, \ldots, x_ky_k)$. We use bold capital letters for matrices as in $\boldsymbol{Z}$. Special case is the $k \times k$ identity matrix, which we denote by $\boldsymbol{I}_k$ and an $k \times k$ matrix of zeros denoted by $\mathbf{0}_{k\times k}$. Regular capital letters are used for sets as in $B = \{x_1, x_2, x_3. \ldots, x_n\}$. Bold-font notation also applies to functions, i.e. $f : \mathbb{R}^n \to \mathbb{R}$ is a scalar function but $\boldsymbol{f} : \mathbb{R}^n \to \mathbb{R}^m$ is a vector function. The notation $P(x_i) = P(x = x_i)$ is adopted to refer to the probability of random variable x having value x_i and $P(y = y_i \mid x = x_j) = P(y_i \mid x_j)$ denotes the conditional probability of $y = y_i$ given $x = x_j$. Also $E(x)$ refers to the expected value of x and $V(x)$ in turn denotes its variance.

2.2 Time series analysis

Madsen [18] defines time series as an observed or measured realisation of an underlying stochastic process. Time series analysis is then a collection of various methods and techniques used to extract information from time series is order to discover the true nature of this hidden process.

For convenience we not only use the term *time series* to refer to a particular realisation, but also to the process behind this realisation. For example, when we model time series $\boldsymbol{y}$, we actually model the underlying stochastic process. Time series $\boldsymbol{y}$ is just one of its realisation and we use it to estimate the particular form and parameters of the equations in the model. Analogically, when we decompose time series into one or more components, we in fact decompose the underlying stochastic process into a number of separate or interconnected processes and model each one of them.

Depending on the frequency at which the observations are recorded, time series can be divided into two distinct groups:

- *Discrete-time series* with observation made at equally spaced points in time. Observation are usually denoted using the subscript notation y_t where $t \in \mathbb{N}$.
- *Continuous-time series* with observations recorded continuously over some time interval. Here the observations are denoted using the function notation $y(t)$ where usually $t \in [0, 1]$.

Depending on the type of observations one makes, both discrete-time and continuous-time series can be further classified into following categories:

- *univariate time series* where we keep track of only one variable and the observations are in the form of y_t for discrete-time or $y(t)$ for continuous-time series;
- *multivariate time series* where we record the values of k variables at the same time, i.e. observations are in the form of a vectors $(y_{t,1}, y_{t,2}, \dots, y_{t,k})$ for discrete-time and $(y_1(t), y_2(t), \dots, y_k(t))$ for continuous-time series.

Further division can be made on the basis of what is observed. Time series analysis can be applied to both real-valued continuous data and discrete numeric data as well as discrete symbolic data (i.e. letters in a language).

In this thesis we focus on real-valued (kW) univariate discrete-time series. For convenience, from now on whenever we use the term time series we specifically refer to real-valued univariate discrete-time series.

Since $\boldsymbol{y}$ is discrete, we use t to refer to the discrete-valued time of $\boldsymbol{y}$ without exceptions and assume $t \in \mathbb{N}$ at all times. An observations recorded at time t then becomes y_t. All observations are real-valued, meaning that $\forall t \in \mathbb{N} : y_t \in \mathbb{R}$. We use vector $\boldsymbol{y}$ to refer to time series composed of observation $y_1, y_2, \dots$ as a whole, i.e. $\boldsymbol{y} = (y_1, y_2, \dots)$. Additionally, vector $\boldsymbol{y}_t$ denotes time series recorded up to time t and $\boldsymbol{y}_{t_1:t_2}$ represents time series observed between time t_1 and t_2, which can be mathematically described by vectors $\boldsymbol{y}_t = (y_1, y_2, \dots, y_t)$ and $\boldsymbol{y}_{t_1:t_2} = (y_{t_1}, y_{t_1+1}, \dots, y_{t_2})$.

In practice it is often important to consider the length of the time interval between consecutive observations y_t and y_{t+1}. We use the term *granularity* or *resolution* to refer to the length of this time interval.

Time series often display periodic fluctuations. For example retail sales tend to peak every year before Christmas. We use the term *seasonality* to refer to these periodic fluctuations and the term *season* to denote the observations recorded during one period. Let also *frequency* denote the length of a season, i.e. number of observations within the season. Throughout this thesis $m \in N$ will denote the frequency of time series. To continue the aforementioned example, retail sales with monthly granularity exhibit seasonality with period of one year and frequency $m = 12$. It is also possible for time series to exhibit multiple seasonalities. In that case we use $m_1, m_2, \dots$ to refer to their respective frequencies.

The main focus of this thesis is time series forecasting. Time series forecasting exploits patterns found in the time series, seasonal or other, to forecast the future behaviour of the underlying process. Mathematically, given time series $\boldsymbol{y}_t$ the aim is to obtain forecasts denoted by $\hat{y}_{t+1|t}, \hat{y}_{t+2|t}, \dots$ The number of forecast to be produced from $\boldsymbol{y}_t$ is referred to as *forecast horizon.* We use $h \in \mathbb{N}$ to denote the forecast horizon. Given time series $\boldsymbol{y}_t$ and forecast horizon h, our focus is then producing accurate forecasts in the form of vector $\hat{\boldsymbol{y}}_{t+h|t}$ defined as

$$\hat{\boldsymbol{y}}_{t+h|t} = (\hat{y}_{t+1|t}, \hat{y}_{t+2|t}, \dots, \hat{y}_{t+h|t}).$$

The accuracy of forecasts is measured by the difference between forecasts and actual observations (see Section 10.1). This differences are called *residuals* and are computed by taking a simple difference $\epsilon_{t+i} = y_{t+i} - \hat{y}_{t+i|t}$ where $i = 1, 2, \dots, h$.

2.3 Deterministic and stochastic time series

Time series analysis is concerned with uncovering stochastic processes that manifests themselves in the form of time series. When a process does not contain any stochastic element and the observation y_{t+1} can be generated by a deterministic algorithm only from $\boldsymbol{y}_t$, the process and time series it generates is said to be *deterministic*. For example time series sampled from sine wave is deterministic. Deterministic time series are also assumed when using a naive method that forecasts by repeating the previous observation (see Section 10.1.4).

On the other hand, when a process does contain stochastic elements, we say that it and by extension its realisations are *stochastic*.

2.4 Stationarity

Time series $\boldsymbol{y}$ (and by extension the underlying process) is said to be stationary if its mean, variation and autocovariance (covariance with itself, specifically with past observations) remain invariant under translation trough time. Expressed mathematically, $\boldsymbol{y}$ is stationary if for all $t, i \in \mathbb{N}$, the following is satisfied:

$$\begin{aligned} E(\boldsymbol{y}_t) &= \mu \\ V(\boldsymbol{y}_t) &= \sigma^2 \\ \operatorname{cov}(y_t, y_{t+i}) &= c(i) \end{aligned}$$

where c is some function.

The stationarity condition is usually violated when dealing with energy consumption data, as there is usually at least one type of seasonality (daily, weekly, annual) and sometimes also trend. There are numerous techniques that one can use to make a time series into a stationary one, i.e. transformations (see Section 10.3.3), deseasonalisation (see Section 10.4) or differencing (see Sections 7.1.4 and 7.1.5.

2.5 White noise process

Many models in time series analysis assume that the process generating time series is composed of a deterministic process and one or more stochastic processes. One of the most common stochastic process considered in such decompositions is the *white noise process*.

White noise is a process whose realisations $\boldsymbol{\epsilon} = (\epsilon_1, \epsilon_2, \dots)$ are generated by repeatedly drawing from normal distribution with zero mean and variance σ^2. Formally, for any $t, i \in \mathbb{N}$ white noise process satisfies the following

$$\begin{aligned} \epsilon_t &\sim \mathcal{N}(0, \sigma^2) \\ E(\boldsymbol{\epsilon}_t) &= 0 \\ V(\boldsymbol{\epsilon}_t) &= \sigma^2 \\ \operatorname{cov}(\epsilon_t, \epsilon_{t+i}) &= 0. \end{aligned}$$

Note that the white noise process is a stationary process.

White noise process $\boldsymbol{\epsilon}$ is usually incorporated into a model by including only the term ϵ_t in model's equations. For convenience, instead of describing the model's equations using " ... where ϵ_t is the value of a white noise process $\boldsymbol{\epsilon}$ at time t" we simply write " ... where $\boldsymbol{\epsilon}$ is a white noise process", even if the term $\boldsymbol{\epsilon}$ is not present in model's equations.

Since residuals of various regression based forecasting models are assumed to be normally distributed [19], it is common practise to use past residuals to generate the white noise process used in the model.

2.6 Lag operator and lag polynomials

In time series analysis we often need a shorthand notation to describe modified time series lagging behind the original. Hamilton [4] suggests the unary *lag operator* L, which takes an observation y_t to produce the previous observation y_{t-1}, mathematically

$$\mathrm{L}\, y_t = y_{t-1}.$$

The definition of L operator can be generalised to produce the next observation

$$\mathrm{L}^{-1}\, y_t = y_{t+1}$$

and also applied repeatedly, which we denote by raising it to the corresponding power:

$$\mathrm{L}^k\, y_t = \mathrm{L}\,\mathrm{L}^{k-1}\, y_t = y_{t-k}.$$

Multiple lag operators can be combined to form a *lag polynomial*. For example, let θ_i be a sequence of coefficients or parameters. Then we can write

$$\theta_0 y_t + \theta_1 y_{t-1} + \cdots = \theta_0\,\mathrm{L}^0\, y_t + \theta_1\,\mathrm{L}\, y_t + \cdots = \sum_{i=0}^{\infty} \theta_i\,\mathrm{L}^i\, y_t = \Theta(\mathrm{L}) y_t$$

where $\Theta(\mathrm{L})$ specifies the lag polynomial with coefficients θ_i.

For all intends and purposes lag polynomials can be multiplied comutatively $\Theta(\mathrm{L})\Phi(\mathrm{L}) = \Phi(\mathrm{L})\Theta(\mathrm{L})$ and divided $\Theta(\mathrm{L})/\Phi(\mathrm{L})$ in the same way as regular polynomials [20].

2.7 Moving average smoothing

Moving average smoothing or smoother is a technique in time series analysis designed to remove noise and better expose the underlying signal [21].

A moving average smoother takes a time series $\boldsymbol{y}$ and computes a new time series $\boldsymbol{y}^* = \mathbf{MAS}_w(\boldsymbol{y})$ whose observations are a result of averaging several observations in the original time series. The observations whose average is used when computing y_i^* are located within a *smoothing window* of length w centred around i, mathematically

$$\mathbf{MAS}_w(\boldsymbol{y})_t = y_t^* = \frac{1}{w} \sum_{i=t-\lfloor w/2 \rfloor}^{t+\lceil w/2 \rceil - 1} y_i \tag{2.1}$$

where $\lfloor w/2 \rfloor$ rounds $w/2$ down to the nearest integer and $\lceil w/2 \rceil$ rounds it up.

The smoothing window w is a parameter of the method and heavily influences the result. The bigger the window the less noisy and more smooth the result is. However, some features (peaks and valleys) of time series that are desirable to preserve may also be smoothed away.

Note that for smoothing windows of odd lengths, the number of observations prior to y_t included in the averaging is the same as the number of observation after y_t. For example, $\mathbf{MAS}_5$ computes y_t^* as $(y_{t-2}+y_{t-1}+y_t+y_{t+1}+y_{t+2})/5$. However, windows of even lengths result in asymmetric averages not centred around t, e.g. $\mathbf{MAS}_4$ computes y_t^* as $(y_{t-2}+y_{t-1}+y_t+y_{t+1})/4$. This can be remedied by taking a moving average with window $w_2 = 2$ after the first moving average with window $w_1 = 4$. The result then looks like

$$y_t^* = \frac{1}{2}\left(\frac{y_{t-2}+y_{t-1}+y_t+y_{t+1}}{4} + \frac{y_{t-1}+y_t+y_{t+1}+y_{t+2}}{4}\right).$$

In literature this type of moving average smoothing is referred to as 2×4 *double moving average smoothing* and we denote this using the following notation:

$$\mathbf{MAS}_{w_2 \times w_1}(\boldsymbol{y}) = \mathbf{MAS}_{w_2}(\mathbf{MAS}_{w_1}(\boldsymbol{y}))$$

Double moving average smoothing can be easily extended to triple etc. moving average smoothing. Double and even triple moving averages are routinely used in time series decomposition to isolate the trend component (see [21], [22], [11] or [23]).

3. Data analysis

In this chapter we focus on the problem of forecasting energy load profiles (Section 3.1), analysis of energy consumption data (Sections 3.2) and weather data (Section 3.3).

3.1 Energy consumption forecasting

Energy consumption forecasting is the practise of estimating the future magnitudes of energy load over a future time period. Accurate forecasts can be utilised by both producers and consumers of electicity [12] in a number of ways, including the following:

- Financial planning: load forecasts can guide executives to make long term revenue projections that are basis for acquisitions, new projects and their budgets, technologies and human resources etc.
- Transmission and distribution: the transmission grid and accompanying systems must be regularly maintained and upgraded to meet the ever-changing demand and improve reliability. Forecasts estimate when and by how much the load will change as well as how the number of customers will grow.
- Demand side management: energy companies can make long term planning according to the forecasts of end-user behaviour. On the other hand, consumers can adjust the schedule of more energy-demanding tasks in a process called load shifting.
- Maintenance: load patterns obtained from forecasts help system operators plan maintenance outages.

Because load forecasting covers such a wide spread of applications, there are many criteria that can be used to distinguish between them. The most important is probably the length of forecast horizon, which segregates load forecasting into following categories:

- *very short term forecasting* deals with forecast horizons from minutes up to a few hours and can be used for example in scheduling of electricity generation [24];
- *short term forecasting* includes forecast horizons measured in days and is the primary focus of demand side management;
- *medium term forecasting* produces forecasts for horizons with length in terms of weeks or a few months and can be used for outage and maintenance planning, as well as load switching operations [25];
- *long term forecasting* uses months, quarters or even years as forecast horizons to for instance develop future generation, transmission, and distribution facilities [26].

When dealing with shorter forecast horizons it is usually sufficient to consider only past observation for relatively accurate predictions. However, as the forecast horizon grows, the amount and the number of sources of information needed for accurate forecasts increases [12], which is summarised in the following:

- very short term forecast only require past loads;
- short term forecasts may require weather information;
- medium term forecasts usually necessitate weather as well as economic information;
- long term forecasts need weather, economic, demographic and sometimes land use information.

Another important criterion is the desired *granularity* or *resolution* of forecasts which retroactively affects the granularity of data suitable used for the analysis. The granularity and forecast horizon are usually interdependent and range from granularity in terms of minutes for very short term forecasting and hourly granularity for short term forecasting to weekly and monthly granularity for medium term forecasting and quarterly or annual granularity for long term forecasting.

There is no consensus regarding what the thresholds separating these categories should be so the divisions presented above should only serve illustrative purposes and may be inconsistent with divisions in some publications.

In this thesis we focus on half-hourly forecasts of energy load profiles one day ahead, i.e. short term forecasts, because the ultimate goal is to use this information in load shifting or cost optimisation. All observations are recorded in kW. Given the history of energy consumption as time series $\boldsymbol{y}$, we define *load profile* for day $d \in \mathbb{N}_0$ as the vector $\boldsymbol{y}_{48d+1:48d+48} = (y_{48d+1}, y_{48d+2}, \dots, y_{48d+48})$. The value y_{48d+1} refers to the average consumption between 00:00 and 00:30, y_{48d+2} contains the average consumption between 00:30 and 01:00 and so on until y_{48d+48}, which represents the average consumption between 23:30 and 24:00. The forecast of load profile for the next day based on $\boldsymbol{y}_t$ is then produced as $\hat{\boldsymbol{y}}_{t+48|t} = (\hat{y}_{t+1|t}, \hat{y}_{t+2|t}, \dots, \hat{y}_{t+48|t})$.

3.2 Energy consumption data

One of the biggest problems related to Smart Grids and energy consumption analysis in particular is the scarcity and low quality of real world energy consumption data. The energy companies are both unwilling and in many cases unable to share the data because energy demands of a customer are considered a private information. Even when this is not the case and the data is available, it is usually of little use because of very low granularity. The reason for this is that because of the current electric grid being designed only for a one-way transfer of energy from supplier to customer, it suffices for an energy company to record only the total amount of energy transferred to a customer during the whole month and bill him accordingly. Naturally, the granularity of one observation per month is nowhere near the requirements of load-shifting optimisation algorithms. Furthermore, the quantity of the data should ideally span several years in order to capture the intrinsic seasonality of energy demand that can be utilised in forecasting.

Therefore, the burden of collecting data suitable for load shifting lies usually on the customer and in many cases a researcher has no choice but to perform the observations himself. As this can be both expensive and time consuming, one may resort to generating energy consumption artificially using some sort of energy demand generator [27]. However, if we train a statistical or machine learning model on artificial data, the model may learn the inner-workings of the algorithm generating the data instead of the real world process behind energy demands of a household. Evaluating the difference in performance of models trained on artificially generated data versus models trained on real-world data may be explored in future work.

Fortunately, from Georges Hebrail and Alice Barard we were able to obtain high-granularity, high-quantity real-world data suitable for analysis [28]. The data comes from a house in Sceaux (92330) which is 10 kilometres south of Paris. The house uses gas-based heating system, has three floors and seven rooms and is inhabited by a family of four or five: two parents working full time and two or three children [29].

Energy consumption data contains 2075259 observations of power consumption in kW sampled as a rate of one observation per minute, i.e. granularity of one minute. The average power consumption per minute is 1.092 kW. The data amounts to just over 1441 days or almost four years of energy consumption. The models in this thesis were trained on data aggregated into half-hour time intervals in a process called *aggregation* described in Section 10.3.1. Half-hour time intervals were chosen because that is the granularity of available weather data, which we describe later in Section 3.3.

One of the problems of this data is that it does not account for energy spent on heating.

Another problem are missing observations. We use the term *outage* to refer to any number of consecutive missing observations, i.e. outage $o = 2$ means that 2 consecutive observations are missing. We also consider outage $o = 0$ representing no missing observations. Figure 3.1 contains a histogram of all outages and their lengths present in the data.

We can see that the total number of missing observations is 25979, which makes up approximately 1.25% of the data or just over 18 days. The missing observations are spread across multiple outages, the longest one lasting for 5 days. In Figure 3.2 we display the length of outages when the data is partitioned into half hour intervals.

Because time series observations depend on previous observation, we cannot simply discard missing observation, but instead fill them with valid values in a process called *imputation* described in Chapter 11. When aggregating minutes into half-hour time intervals, the process may discard valid observations in time intervals containing only a few missing observation. For this reason we first impute the data and then perform aggregation.

From now on we analyse imputed (Chapter 11) and aggregated (Section 10.3.1) data with one observation per half-hour time interval.

As the histogram in Figure 3.3 shows, energy consumption approximately follows a bimodal distribution with peaks around 0.35 kW and 1.4 kW.

Figure 3.4 is the result of averaging daily energy consumption profiles across the whole datasets. A clear pattern with morning and evening peaks emerges. The

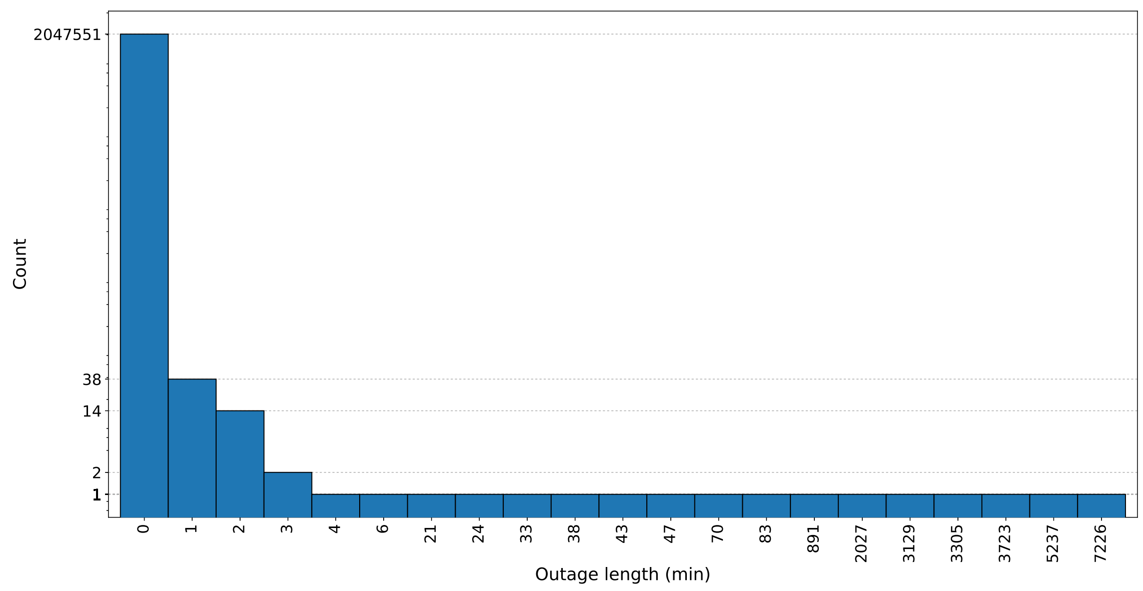

Figure 3.1: Histogram of the number of outages of different lengths

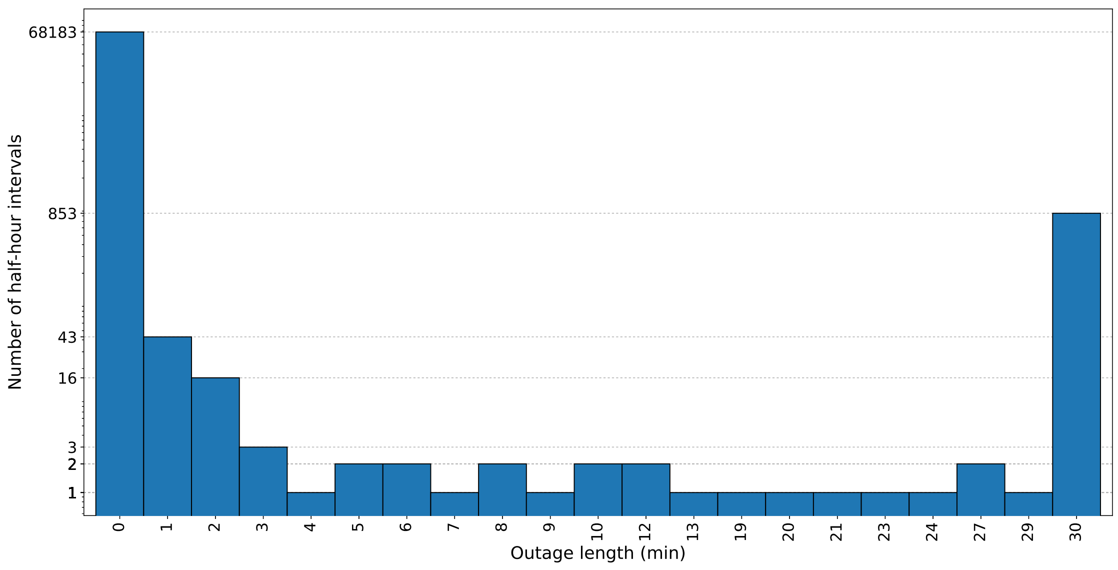

Figure 3.2: Histogram of the number of outages of different lengths when the data is partitioned into half-hour intervals, each half-hour interval contains an outage $o = 0, 1, \ldots, 30$ illustrating how many minutes of that time interval are in fact missing

smaller morning peak between 7:30 and 8:00 is probably the result of parents and children waking up and getting ready for school and work. Then the load decreases while the residents are away from home before peaking again between 20:30 and 21:00. This evening peak is probably caused by the necessity of artificial lighting and various devices used during leisure activities like television and computers.

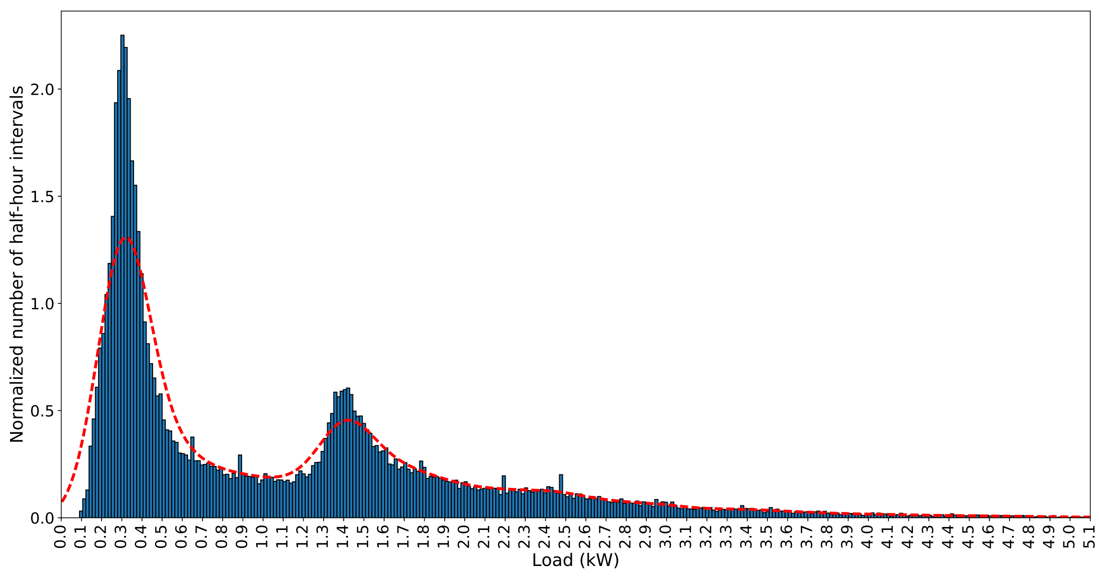

Figure 3.3: Histogram of energy loads during half-hour time intervals, red line represents the bimodal distributions that we fit to the data

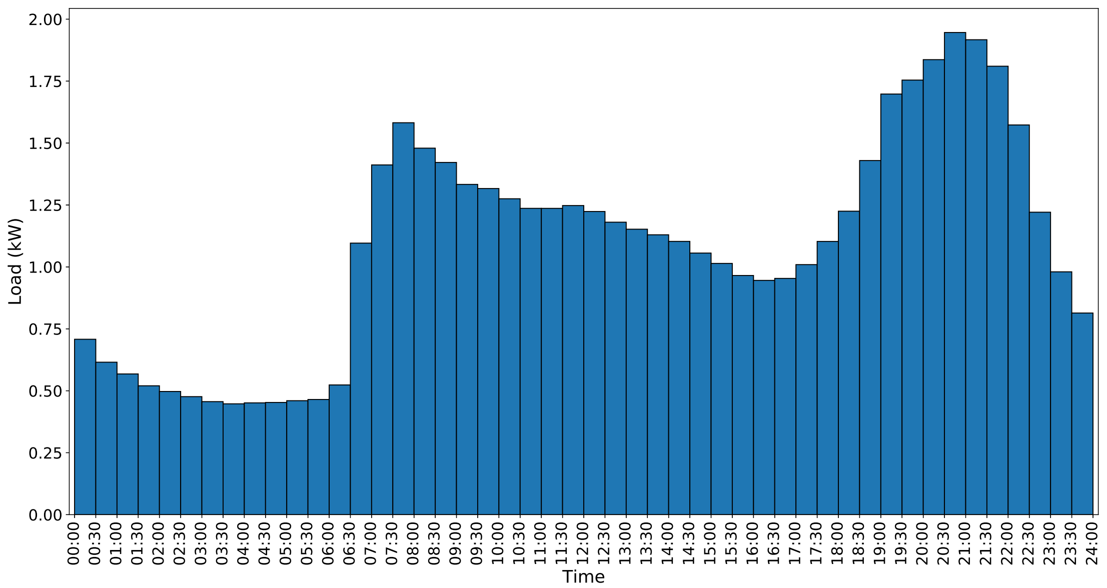

Figure 3.4: Average load profile.

During night, especially after midnight the power consumption is at its lowest while the inhabitants are asleep and the next day the cycle starts anew. This periodicity hints at daily seasonality: the same pattern repeated day after day.

Since business days in France span from Monday to Friday, it is natural to expect energy consumption profiles during those days to be more stable and predictable when compared to weekends. This is illustrated in Figure 3.5. Notice that while evening peak is still prominent each day, the morning peak is all but

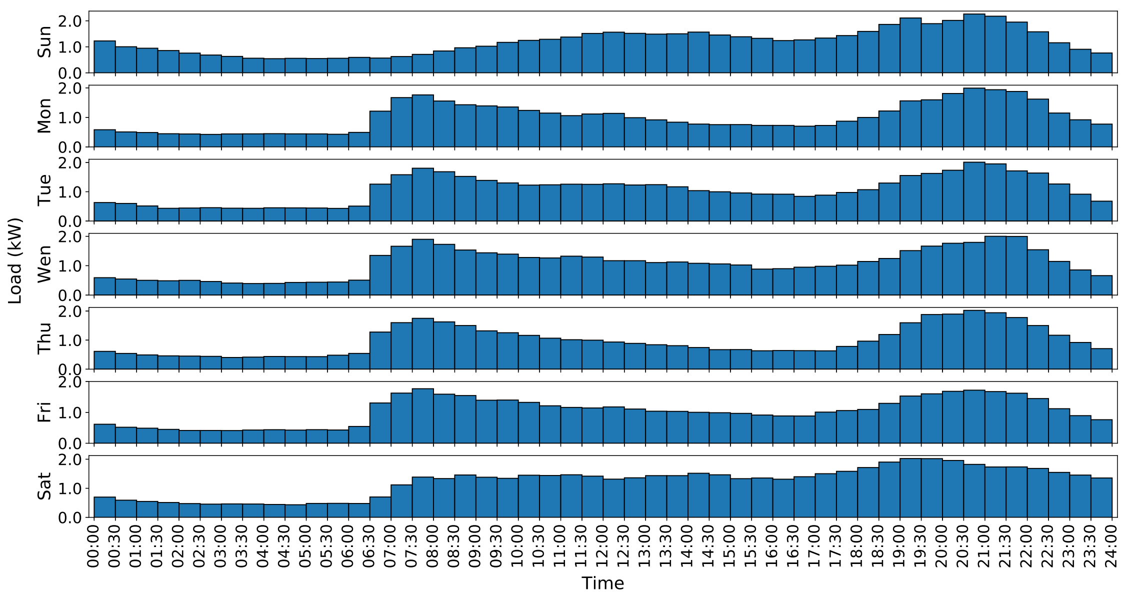

Figure 3.5: Average load profile for each day of the week

absent on Sunday and Saturday. Instead the energy consumption gradually picks up as the day progresses with a plateaux around noon. Also the evening peak on Saturday occurs sooner than on the other days. The night between Saturday and Sunday exhibits unusually high energy consumption when compared to school nights as people are usually more prone to staying up late during weekend. We conclude that business days can differ from weekends significantly. The differences may also manifest themselves in some kind of weekly seasonality.

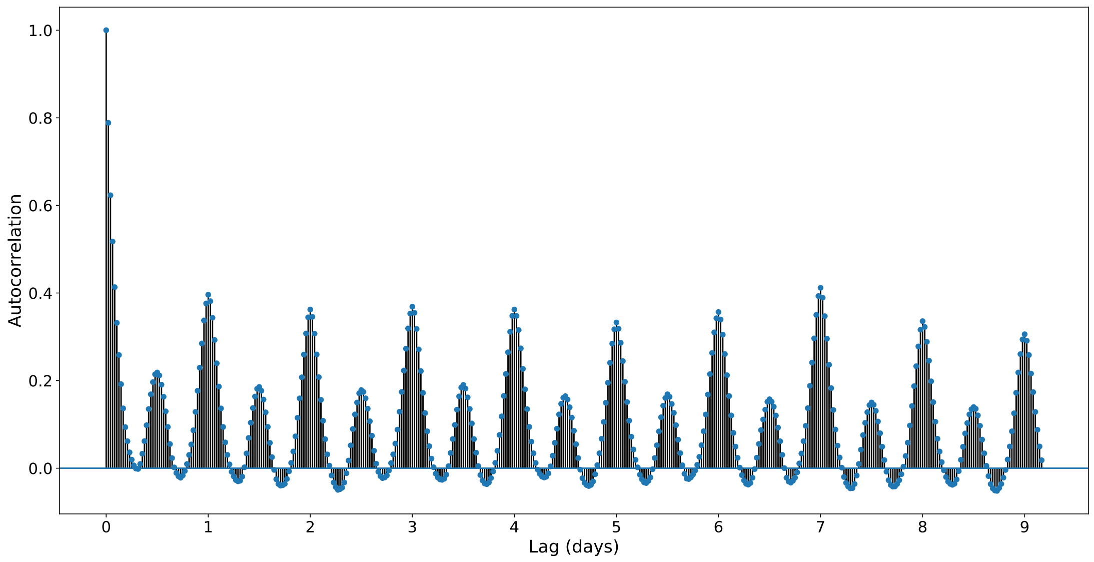

Figure 3.6: Autocorrelation with the previous nine days, one day corresponds to data lagged by 48 half-hours or 24 hours

One possible way of discovering patterns representing different types of seasonality in our data is the *autocorrelation function* (ACF). ACF function of time series is a correlation of time series as a function of its own lagged values. Mathematically, it is defined as

$$\mathrm{ACF}_t(i) = \frac{E((y_t - \mu_t)(y_{t-i} - \mu_{t-i}))}{\sigma_t \sigma_{t-i}} \tag{3.1}$$

where μ_t, μ_{t-i} are means and σ_t, σ_{t-i} are standard deviations of time series $\boldsymbol{y}_t$ and $\boldsymbol{y}_{t-i}$ respectively.

Figure 3.6 displays the autocorrelation of our data plotted against lags as high as nine days. Naturally the highest correlation arises between consecutive observations. Then the ACF quickly plummets hitting correlations close to zero for lags around six hours. There are multiple types of seasonality apparent from Figure 3.6. The most noticeable is daily seasonality with peaks every 24 hours. More prominent however is weekly seasonality, which is also confirmed by the results of naive forecasting method in Section 10.1.4. Then there is also the case of 12-hour seasonality. Nevertheless, as our task is to produce forecasts one day ahead, we are unable to utilise 12-hour seasonality.

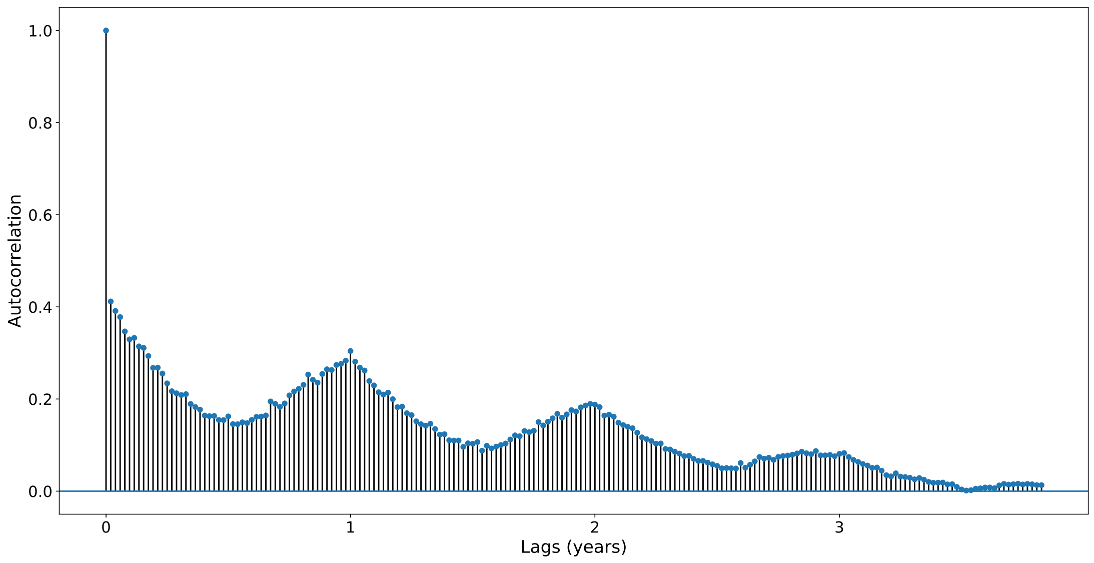

Figure 3.7: Autocorrelation, only weekly lags displayed, one year is equal to 52 weeks

Figure 3.7 contains ACF plot for all lags, i.e. as many as there are observation (2075259). For clarity we included only weekly lags. It is clear that there is also annual seasonality present in the data, because ACF peaks around lags corresponding to 52, 104 and 156 weeks.

From this we can conclude that our data contains multiple types of seasonality including daily, weekly and annual. However, it is important to note that AFC only captures linear relationship between lagged values and may possibly miss some type of nonlinear relationship that may also be exploited by suitable models, e.g. neural networks.

3.3 Weather data

One of the main factors affecting energy consumption is the outside weather conditions [30]. For this reason we collected historical weather data using *Underground Weather API* [31]. The weather data comes from Orly airport near Paris, which lies eight kilometres from the house in question and is the nearest meteorological station [29]. Various weather characteristics are sampled at a rate of one observation per 30 minutes.

From among many various weather characteristics provided by the Underground weather API, we chose temperature (°C), relative humidity and wind speed (km/h). We were unable to obtain solar irradiation from the API. The criteria for choosing these characteristics were small number of missing observations, the highest absolute correlation with energy consumption data from among all available characteristics (-0.18 for temperature, 0.057 for wind speed and 0.055 for humidity) and small correlation among these variables (see Table 3.1)

Table 3.1: Correlation of selected weather characteristics

characteristic	temperature (°C)	humidity (%)	wind speed (km/h)
temperature (°C)	1.000000	-0.592530	0.081844
humidity (%)	-0.592530	1.000000	-0.207473
wind speed (km/h)	0.081844	-0.207473	1.000000

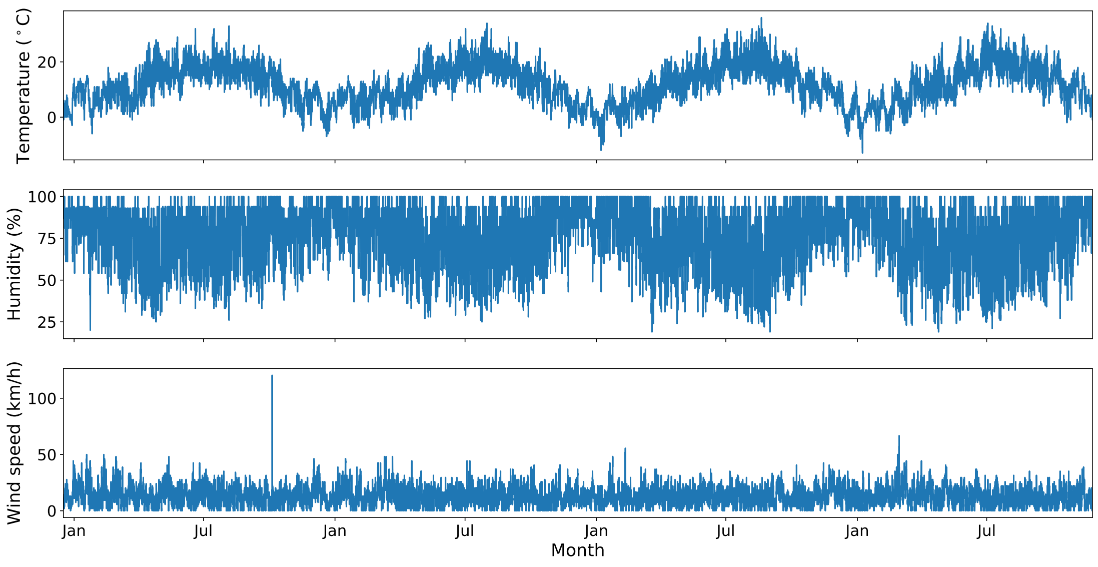

Figure 3.8: Weather data across the years

Figure 3.8 illustrates how these characteristics develop through time. As we can see the temperature rises and falls as expected with colder winters and warmer summers. The humidity indicates drier summers and humid winters. Wind speed resembles white noise.

It is also important to note that using real weather data to produce forecasts may introduce unwanted bias. Ideally, one should opt for using historical forecasts of weather data instead of real weather data. However, we were unable to obtain historical weather forecasts and were thus left with no choice.

4. Literature overview

In this chapter we present previous studies examining a variety of both statistical and machine learning methods applied to load forecasting.

Papalexopoulos and Hesterberg [32] examined the performance of a regression based model on data collected by Pacific Gas and Electric Company in California. The data for the estimation of the model included energy consumption and real historical temperature (not historical forecasts) sampled at a rate of one observation per hour. Authors also demonstrate that contaminating the historical weather data by a random noise in order to simulate historical forecasts results in a less accurate model. In addition to energy consumption and temperature, the regression model incorporated daylight saving time and holidays in the form of binary variables. One-day-ahead forecasts (forecast horizon $h = 24$) were produced every midnight. The model performed with an average error of 12 MW.

A different model based on the Holt-Winters exponential smoothing was used in a study conducted by Taylor [33]. The data consisted of half-hourly observations of energy consumption and forecasts are produced for forecast horizons ranging from one half-hour ($h = 1$) to one day ($h = 48$). Taylor identified two types of seasonality that are present in the data: daily and weekly. In order to accommodate the second type of seasonality, Holt-Winters method was modified and compared to both traditional Holt-Winters method and ARIMA model. According to mean absolute percentage error (MAPE), ARIMA model performed the best. However, after fitting an additional AR model to the residuals of the modified Holt-Winters method, the accuracy of forecasts surpassed that of ARIMA for all forecast horizons. For even more accurate forecasts, Taylor recommends combining several different methods in an ensemble.

Pappas et al. [34] conducted a study of ARMA models on energy consumption data provided by Hellenic Public Power Corporation in Greece. The data was sampled at a rate of one observation per day. Authors identified weekly and annual seasonalities and removed them both through deseasonalisation before fitting the model. Forecast were produced one day (forecast horizon $h = 1$) and one week (forecast horizon $h = 7$) ahead. Paper also examines various criteria used in estimation of ARMA models, namely the AIC, AICC, BIC and MMPF, and their impact on the model's performance. Authors concluded that the best criterion was MMPF with MAPE of 1.87% followed by AICC with 1.98%.

Tassou and Marriot [35] studied the ability of neural networks to predict the electricity consumption in a supermarket. They were especially interested in identifying the most important inputs for prediction and also comparing neural networks with multiple regression techniques. The power consumption in a supermarket is recorded every half-hour and together with environmental conditions is used to train the network. Forecasts are made for an undisclosed number of half-hours ahead. Neural networks with correlation coefficient (R2) of 95% outperform regression analysis with 79% in this situation. The authors also pinpointed time of day as the most important factor in determining energy consumption.

Kalogirou and Bojic [36] proposed a neural network to predict the energy consumption of a passive solar building. In the design of a passive solar house, the windows, walls and floors are built in such a way as to collect, distribute and store solar energy in the winter and reflect it in the summer. Passive solar buildings differ from active solar buildings in that they do not use any mechanical or electrical devices like boilers and pumps in their heating system [37]. Instead they optimise characteristics such as size and placement of windows, shading, insulation, thermal mass and glazing type. Authors' objective was to build a simulation software based on neural networks in order to model the thermal behaviour of the building, because a trained neural networks is faster than a traditional physical model based on differential equations. A physical model called ZID, developed by Energy Management Centre of the University of Kragujevac, was used to generate training samples of twelve hourly energy loads per day for a summer and a winter season. This data was then used to train the network. Authors examined a number of different types of neural networks with varying number of layers. Paper concludes with presenting the result of a selected type of network called Jordan Elman recurrent network, which was able to reach R2 value of 0.9991.

A different approach was proposed by Sulaiman, Jeyanthy and Devaraj [38]. Using data from Smart Meter they forecast hourly load for a day as a whole. Smart Meters are able to provide high resolution data every few seconds. These were then used to train neural networks with varying number of neurons. Data was sampled at a rate of 24 observations per day and forecast were made one day ahead, i.e. with forecast horizon $h = 24$. For the evaluation of forecasts authors used hit rate as the accuracy measure. Forecast for a particular hour was considered a hit if fell within ±10% of a true value that is above 1 kW and within ±100 W if it was below 1 kW. The best network achieved hit rate of 70.54%. The paper also contains a comparison of hit rate of the network with respect to hour of the day. It performed best in the night-time with little or no human interference.

Another comparison was conducted in a paper by Neto and Fiorelli [39]. Using data from the Administration Building of the University of São Paulo in Brazil, the authors compared the the accuracy of neural networks with that of a physical model called EnergyPlus [40]. Energy consumption data was sampled at a rate of one observation per hour and authors forecast one day ahead only for business days (forecast horizon $h = 24$). Authors also performed parameter analysis to assess the significance of individual factors for prediction. Their paper focuses on two types of neural networks. The first was a simple network with only temperature as its input, the second takes temperature, relative humidity and also solar radiation into account. Moreover, different networks were used for business days and weekends. Authors concluded that both neural networks and EnergyPlus are suitable because for 80% of samples the absolute error was within 10% and 13% respectively. They identified external temperature, internal heat gains and equipment performance as the most significant factors, while humidity and solar radiation had negligible effects.

5. State-space models

Originating in control engineering, state-space models are mathematical models of physical systems that vary through time. In state-space models, a physical system is thought of as having intrinsic unobservable state that changes as the system evolves. Although the state is unobservable, it manifests itself as a sequence of measurable quantities, usually contaminated by noise, that can be represented as time series. State-space model is then a number of equations that capture both the particularities of this manifestation as well as the evolution of the system's state. For example, for a physical system consisting of a satellite orbiting the Earth, the intrinsic state consists of velocity, angular momentum, mass, atmospheric drag and possibly other quantities, while from Earth only its position in orbit, surely contaminated by noise, is observable. Throughout this chapter we use s as the number of these hidden quantities. State-space model can help estimate either past, current or future positions based on a sequence of observations and a sequence of estimates of the satellite's state.

This chapter presents an overview of the theory around state-space models published mainly in book by Hamilton [4], Chatfield [41], Hyndman et al. [11], Libert et al. [42] and Durbin and Koopman [43]. Its purpose is to serve as a foundation for Chapters 6, 7 and 8, where the theory is put into practice.

The chapter is divided into five sections. In Section 5.1 the general form of state-space model is introduced. Section 5.2 and Section 5.3 narrow the general definition of state-space models to better suit the purpose of this thesis. While describing the state-space models in these three sections we assume that various parameters in their equations are known. The process of estimating these parameters is presented in Section 5.4. Lastly, in Section 5.5 we describe how state-space models are used in forecasting time series.

State-space models are able to accommodate continuous-time series and also multivariate time series. However, for our purposes it is enough to consider only state-space models for the case of univariate discrete-time series.

5.1 General state-space models

The basic idea behind state-space models is that at any time the measurement of a signal is contaminated by noise, which can be intuitively expressed as:

$$observation = signal + noise. \tag{5.1}$$

In state-space models the signal at time t is considered to be a combination of a set of variables, called *state variables*. State variables are collected together to form a *state vector* or simply *state*.

Let us consider an univariate time series $\boldsymbol{y}$ and let $\boldsymbol{x}_t \in \mathbb{R}^s$ be a state vector at time t with s as the number of state variables. Then we may rewrite (5.1) as the so-called *observation equation*

$$y_t = w_t(\boldsymbol{x}_{t-1}) + r_t(\boldsymbol{x}_{t-1})e_t \tag{5.2}$$

where e_t represents the observation error or noise and $w_t, r_t : \mathbb{R}^s \to \mathbb{R}$ are assumed to be known scalar functions. Function w_t describes how the state variables are combined to produce the observation and function r_t can be interpreted as the effect the noise has on these state variables.

State-space models postulate that the state vectors satisfy the *Markov property*:

$$\forall t \in \mathbb{N} : P(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}, \boldsymbol{x}_{t-2}, \dots, \boldsymbol{x}_0) = P(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}).$$

This means that the future behaviour of the system is completely determined only by the most recent values of state variables.

Since it is not always possible to observe the elements of the state vector $\boldsymbol{x}_t$ directly, state-space models make the assumption that the state vector evolves according to the so called *state equation*

$$\boldsymbol{x}_t = \boldsymbol{f}_t(\boldsymbol{x}_{t-1}) + \boldsymbol{g}_t(\boldsymbol{x}_{t-1}) \odot \boldsymbol{\xi}_t \tag{5.3}$$

where $\boldsymbol{f}_t, \boldsymbol{g}_t : \mathbb{R}^s \to \mathbb{R}^s$ are vector function assumed to be known and $\boldsymbol{\xi}_t \in \mathbb{R}^s$ is a vector of disturbances. Function $\boldsymbol{f}_t$ describes the process of transforming the previous state-space vector $\boldsymbol{x}_{t-1}$ into the current one $\boldsymbol{x}_t$ and function $\boldsymbol{g}_t$ explains how are each of the state variables affected by the noise.

The observation (5.2) and state (5.3) equations together with specified functions $w_t, r_t, \boldsymbol{f}_t, \boldsymbol{g}_t$ and the distribution of the error term e_t and disturbances $\boldsymbol{\xi}_t$ are what constitute the *general state-space model.*

In the most general case, $w_t, r_t, \boldsymbol{f}_t$ and $\boldsymbol{g}_t$ are subject to change in time. However, often this is not the case and they can be assumed to be constant with respect to t. They are then said to be *time-invariant* and we can replace them by $w, r, \boldsymbol{f}, \boldsymbol{g}$ in observation (5.2) and state (5.3) equations.

5.2 Innovations state-space models

Consider a state-space model with observation equation (5.2) and state equation (5.3) in the following form:

$$y_t = w_t(\boldsymbol{x}_{t-1}) + r_t(\boldsymbol{x}_{t-1})\epsilon_t \tag{5.4}$$
$$\boldsymbol{x}_t = \boldsymbol{f}_t(\boldsymbol{x}_{t-1}) + \boldsymbol{g}_t(\boldsymbol{x}_{t-1})\epsilon_t \tag{5.5}$$

where $w_t, r_t : \mathbb{R}^s \to \mathbb{R}$, $\boldsymbol{f}_t, \boldsymbol{g}_t : \mathbb{R}^s \to \mathbb{R}^s$ and $\boldsymbol{\epsilon}$ is a white noise process.

In this state-space model all disturbances $\boldsymbol{\xi}_t$ are modelled using the same white noise process $\boldsymbol{\epsilon}$, which means that all sources of error now have the same origin. Because $\boldsymbol{\epsilon}$ represents what is new and unpredictable, it is sometimes called an *innovation* and state-space model having these innovations as the single source of randomness is therefore an *innovations state-space model.*

5.3 Linear state-space models

Linear state-space models are a special case of general state-space models in that they assume that the observation equation (5.2) and state equation (5.3) can be expressed as a linear combination of state variables:

$$\begin{aligned} y_t &= \boldsymbol{w}_t'\boldsymbol{x}_{t-1} + e_t \\ \boldsymbol{x}_t &= \boldsymbol{F}_t\boldsymbol{x}_{t-1} + \boldsymbol{g}_t \odot \boldsymbol{\xi}_t \end{aligned}$$

where $\boldsymbol{w}_t, \boldsymbol{g}_t \in \mathbb{R}^s$, $\boldsymbol{F}_t \in \mathbb{R}^{s\times s}$.

This is a general state-space model with constant $r_t(\boldsymbol{x}_{t-1}) = 1$, constant $\boldsymbol{g}_t(\boldsymbol{x}_{t-1})$ with respect to $\boldsymbol{x}_t$ and linear functions $f_t(\boldsymbol{x}_{t-1}) = \boldsymbol{F}_t\boldsymbol{x}_{t-1}$, $w_t(\boldsymbol{x}_{t-1}) = \boldsymbol{w}_t'\boldsymbol{x}_{t-1}$.

In this thesis the majority of state-space models can be expressed as *time-invariant linear innovations state-space models*, which can be described using the observation and state equations

$$y_t = \boldsymbol{w}'\boldsymbol{x}_{t-1} + \epsilon_t \tag{5.6}$$
$$\boldsymbol{x}_t = \boldsymbol{F}\boldsymbol{x}_{t-1} + \boldsymbol{g}\epsilon_t \tag{5.7}$$

where $\boldsymbol{w}, \boldsymbol{g} \in \mathbb{R}^s$, $\boldsymbol{F} \in \mathbb{R}^{s\times s}$ and $\boldsymbol{\epsilon}$ is a white noise process.

Therefore, for convenience, when we use the term "state-space model" in other chapters, we specifically refer to time-invariant linear innovations state-space models described by observation equation (5.6) and state equation (5.7). Any deviations from this convention will be explicitly stated.

5.4 Estimation

One of the greatest advantages of the statistical theory behind state-space models is that the methods of statistical inference can be used to estimate the parameters of the models. Since the majority of state-space models in this thesis are linear innovations state-space models, these parameters are $\boldsymbol{w}_t, \boldsymbol{F}_t$ and $\boldsymbol{g}_t$.

To estimate these parameters we use the maximum likelihood estimation. In the context of state-space models, maximum likelihood estimation is an optimisation algorithm that maximises the "likelihood" of a time series $\boldsymbol{y}$ arising from a particular model with respect to its parameters. This is described in Section 5.4.2. To compute this "likelihood" the majority of state-space models in this thesis use Kalman filter, see Section 5.4.1.

5.4.1 Kalman Filter

First introduced by Kalman in 1960 [44], Kalman filter is a method for estimating system's varying state. It improved upon other methods of the time by incorporating system's state-space representation and multiple sequential observations to form an estimate that is better than the estimate obtained using only one observation.

All state-space models in this thesis that use Kalman filter are linear innovations state-space models. Therefore, in this section we present a version of Kalman filter adapted for this particular type of state-space models, but everything that follows can be restated also for a general state-space model. Linear innovations state-space model is, as the name suggests, a combination of linear and innovations state-space model. It can be expressed using the following observation and state equations:

$$y_t = \boldsymbol{w}_t'\boldsymbol{x}_{t-1} + \epsilon_t$$
$$\boldsymbol{x}_t = \boldsymbol{F}_t\boldsymbol{x}_{t-1} + \boldsymbol{g}_t\epsilon_t$$

where $\boldsymbol{x}_t \in \mathbb{R}^s$ is the state vector containing s state variables, $\boldsymbol{w}_t, \boldsymbol{g}_t \in \mathbb{R}^s$, $\boldsymbol{F}_t \in \mathbb{R}^{s \times s}$ and $\boldsymbol{\epsilon}$ is a white noise process.

For the distributions of $\boldsymbol{y}$ and $\boldsymbol{x}_t$ Durbin [43] proves that the following holds:

$$P(y_t \mid \boldsymbol{x}_{t-1}, \boldsymbol{x}_{t-2}, \dots, \boldsymbol{x}_0, \boldsymbol{y}_{t-1}) = P(y_t \mid \boldsymbol{x}_{t-1})$$
$$P(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1}, \boldsymbol{x}_{t-2}, \dots, \boldsymbol{x}_0, \boldsymbol{y}_{t-1}) = P(\boldsymbol{x}_t \mid \boldsymbol{x}_{t-1})$$

Let $\hat{\boldsymbol{x}}_t = E(\boldsymbol{x}_t \mid \boldsymbol{y}_{t-1})$ be the state estimate and $\boldsymbol{V}_t = V(\boldsymbol{x}_t \mid \boldsymbol{y}_{t-1})$ be the state variance. The assumption of Kalman filter is that given $\boldsymbol{y}_{t-1}$ the state vector $\boldsymbol{x}_t$ is from normal distribution $\boldsymbol{x}_t \sim \mathcal{N}(\hat{\boldsymbol{x}}_t, \boldsymbol{V}_t)$, where for the initial state $\boldsymbol{x}_0 \sim \mathcal{N}(\hat{\boldsymbol{x}}_0, \boldsymbol{V}_0)$ both $\hat{\boldsymbol{x}}_0$ and $\boldsymbol{V}_0$ are assumed to be known.

The objective of Kalman filter is to obtain $\hat{\boldsymbol{x}}_t$ and $\boldsymbol{V}_t$ from the previous estimates $\hat{\boldsymbol{x}}_{t-1}$ and $\boldsymbol{V}_{t-1}$ so that it minimises $\sum_{t=1}^{n}(\hat{\boldsymbol{x}}_t - \boldsymbol{x}_t)^2$. This is done by iterating the following recursions:

$$r_t = y_t - \boldsymbol{w}_t' \hat{\boldsymbol{x}}_{t-1} \tag{5.8}$$
$$\boldsymbol{k}_t = q_{t-1}^{-1} \boldsymbol{F}_t \boldsymbol{V}_{t-1} \boldsymbol{w}_t \tag{5.9}$$
$$\hat{\boldsymbol{x}}_t = \boldsymbol{F}_t \hat{\boldsymbol{x}}_{t-1} + r_t \boldsymbol{k}_t \tag{5.10}$$
$$q_t = \boldsymbol{w}_t' \boldsymbol{V}_{t-1} \boldsymbol{w}_t + \sigma^2 \tag{5.11}$$
$$\boldsymbol{V}_t = \boldsymbol{F}_t \boldsymbol{V}_{t-1} (\boldsymbol{F}_t - \boldsymbol{k}_t \boldsymbol{w}_t')' + \sigma^2 \boldsymbol{g}_t \boldsymbol{g}_t' \tag{5.12}$$

where σ^2 is the variance of $\boldsymbol{\epsilon}$, $\boldsymbol{k}_t \in \mathbb{R}^s$ and $\boldsymbol{V}_t \in \mathbb{R}^{s \times s}$.

Vector $\boldsymbol{k}_t$ is called *Kalman gain* and it describes relative certainty of the observations y_t and current state estimate $\hat{\boldsymbol{x}}_t$. A higher gain represents high confidence in the current estimates, which results in more weight placed on recent observations and a more responsive filter. Low gain disregards the recent observations and follows the state estimates $\hat{\boldsymbol{x}}_t$, effectively smoothing out noise at the cost of responsiveness. Kalman gain depends on two other parameters, q_t and $\boldsymbol{V}_t$.

q_t is the estimated covariance of residuals r_t that represents the variability in observation y_t. If q_t is large then y_t changes a lot and consequently the confidence in y_t is small which is represented by q_t^{-1}. As a result, the Kalman gain is lower and the filter follows $\hat{\boldsymbol{x}}_t$. On the other hand, for small q_t the filter gravitates towards new observations.

$\boldsymbol{V}_t$ is the state variance which represents the variability of state $\boldsymbol{x}_t$. If it is large, the state is estimated to change wildly, which needs to be reflected in $\hat{\boldsymbol{x}}_t$ by a higher value of the Kalman gain. In case of low $\boldsymbol{V}_t$ the filter tracks new observations more closely.

In the case of time-invariant state-space models, the state variance $\boldsymbol{V}_t$ converges to a constant matrix $\boldsymbol{V}$ [45], which is the solution to a type of discrete-time algebraic Riccati equation:

$$\boldsymbol{V} = \boldsymbol{F}\boldsymbol{V}\boldsymbol{F}' - q^{-1}\boldsymbol{F}\boldsymbol{V}\boldsymbol{w}\boldsymbol{w}'\boldsymbol{V}\boldsymbol{F}' + \sigma^2 \boldsymbol{g}\boldsymbol{g}'$$

where $q = \boldsymbol{w}'\boldsymbol{V}\boldsymbol{w} + \sigma^2$.

This results in considerable computational savings as the state covariance $\boldsymbol{V}_t$ in (5.12) and q_t in (5.11) no longer need calculating and the Kalman filter can be reduced to the following:

$$r_t = y_t - \boldsymbol{w}'\hat{\boldsymbol{x}}_{t-1}$$
$$\hat{\boldsymbol{x}}_t = \boldsymbol{F}\hat{\boldsymbol{x}}_{t-1} + r_t q^{-1} \boldsymbol{F}\boldsymbol{V}\boldsymbol{w}$$

5.4.2 Maximum likelihood estimation

Consider a sample $\boldsymbol{z}_t = (z_1, z_2, \ldots, z_t)$ of t observations from a distribution with an unknown probability density function $f(z_i)$. The main assumption of maximum likelihood estimation is that $f(z_i)$ comes from a *parametric model*, a family of probability distribution functions $\{f(z_i \mid \boldsymbol{\theta}) : \boldsymbol{\theta} \in \boldsymbol{\Theta}\}$ parametrised by vector $\boldsymbol{\theta} \in \mathbb{R}^k$ from $\boldsymbol{\Theta} \subseteq \mathbb{R}^k$ [46]. Mathematically, $f(z_i) = f(z_i \mid \boldsymbol{\theta}_0)$ for some $\boldsymbol{\theta}_0 \in \boldsymbol{\Theta}$. The objective is to find parameters $\hat{\boldsymbol{\theta}}$ as close to $\boldsymbol{\theta}_0$ as possible.

The "closeness" of $\boldsymbol{\theta}$ and $\boldsymbol{\theta}_0$ is evaluated using the *likelihood function* defined as

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{z}_t) = f(\boldsymbol{z}_t \mid \boldsymbol{\theta})$$

where $f(\boldsymbol{z}_t \mid \boldsymbol{\theta})$ is the *joint probability density function* of $f(z_1 \mid \boldsymbol{\theta}), f(z_2 \mid \boldsymbol{\theta}), \ldots, f(z_t \mid \boldsymbol{\theta})$. In essence $\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{z}_t)$ describes how probable it is to obtain $\boldsymbol{z}_t$ from distribution $f(z \mid \boldsymbol{\theta})$.

Maximum likelihood estimations is then an optimisation algorithm tasked with maximizing the likelihood function $\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{z}_t)$ with respect to parameters $\boldsymbol{\theta}$. The resulting $\hat{\boldsymbol{\theta}}$ that best fit the data is obtained from

$$\hat{\boldsymbol{\theta}} = \arg\max_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{z}_t)).$$

The popularity of maximum likelihood estimation is due to the fact that it is *consistent* [47], which means that provided large enough sample $\boldsymbol{z}_t$, it is possible to approximate the parameters $\boldsymbol{\theta}_0$ with arbitrary precision, i.e.

$$\lim_{t \to \infty} \hat{\boldsymbol{\theta}} = \boldsymbol{\theta}_0.$$

Let $\boldsymbol{\theta}$ be a collection of all the parameters of a linear innovations state-space model, i.e. $\boldsymbol{w}_t, \boldsymbol{F}_t, \boldsymbol{g}_t, \boldsymbol{x}_0$ including the variance of white noise σ^2. Furthermore let $\boldsymbol{x}_0 \sim \mathcal{N}(\hat{\boldsymbol{x}}_0, \boldsymbol{V}_0)$ where both $\hat{\boldsymbol{x}}_0$ and $\boldsymbol{V}_0$ are known. Since the Markov property applies for state-space models, the joint probability density function $f(\boldsymbol{y}_t \mid \boldsymbol{\theta})$ can be expressed in the form of

$$f(\boldsymbol{y}_t \mid \boldsymbol{\theta}) = P(\boldsymbol{y}_t \mid \boldsymbol{\theta}) = \prod_{i=1}^{t} P(y_i \mid \boldsymbol{y}_{i-1}, \boldsymbol{\theta})$$

and consequently

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{y}_t) = \prod_{i=1}^{t} P(y_i \mid \boldsymbol{y}_{i-1}, \boldsymbol{\theta}).$$

In the case of linear innovations state space models taking the logarithm of the previous equation translates into

$$\log \mathcal{L}(\boldsymbol{\theta}; \boldsymbol{y}_t) = -\frac{1}{2}\left(t \log 2\pi + \sum_{i=1}^{t} \log |q_i| + \sum_{i=1}^{t} q_i^{-1} r_i' r_i\right)$$

where q_i and r_i are routinely computed by the Kalman filter.

5.5 Smoothing, filtering and forecasting

Let us consider a linear innovations state-space model where we define $\hat{y}_{k|t} = E(y_k \mid \boldsymbol{y}_t)$ as the conditional expectation of y_k given $\boldsymbol{y}_t$. The process of evaluating $\hat{y}_{k|t}$ is called *smoothing* for $k < t$, filtering for $k = t$ and forecasting for $k > t$. Because our task is forecasting and because smoothing can introduce bias to the time series, in this thesis we focus only on filtering and forecasting.

Filtering concerns updating the state vector to reflect new observations. It is done using Kalman filter by doing additional iteration of Equations (5.8)-(5.12) using the new observation. We use filtering in cross-validation described in Section 10.2 (the motivation is to lower the computational time of cross-validation: after we forecast load profile for the next day, the model is filtered on the real profile for that day without re-estimation).

Generaly, in forecasting we want to estimate the value of y_{t+h} from $\boldsymbol{y}_t$. In state-space models the point forecasts are obtained as conditional expectation $\hat{y}_{t+h|t} = E(y_{t+h} \mid \boldsymbol{y}_t)$. According to Durbin [43], the point forecasts for $\hat{\boldsymbol{y}}_{t+h|t} = (\hat{y}_{t+1|t}, \hat{y}_{t+2|t}, \ldots, \hat{y}_{t+h|t})$ can be obtained by iterating the respective observation and state equations of the model for $i = 1, 2, \ldots, h$ while in each step letting $\hat{y}_{t+i|t} = \boldsymbol{w}'_{t+i}\boldsymbol{x}_{t+i-1}$ and $\epsilon_{t+i} = 0$.

6. Exponential smoothing

Exponential smoothing (ES) is a popular technique used in time series forecasting [43]. Originally, it was closely related to the moving average method (not to be confused with moving average process), which in essence assigns equal weights $\frac{1}{k}$ to k past observations. ES modifies this idea by assuming that the more recent the observation, the more relevant the information it carries. This is modelled by a weighting scheme that pairs more recent observations with larger weights and as the distance between observations grows, the weights become exponentially smaller.

More advanced ES methods postulate that the time series $\boldsymbol{y}$ can be decomposed into one, two or three separate components: *level* $\boldsymbol{l}$, *trend* $\boldsymbol{b}$ and *seasonality* $\boldsymbol{s}$. As it turns out, any ES method presented in this chapter can be represented as a state-space model whose state consists of one or more of these components.

The theory and notation in this chapter is borrowed mainly from books by Hyndman et al. [11] and Hyndman with Athanasopoulos [48]. The chapter is divided into three sections. In Section 6.1 we present a number of ES methods of increasing complexity, the motivation behind and distinctions between them. Section 6.2 offers a different perspective of looking at ES, one that treats them as a special case of state space models. We show that underneath any ES method lies a state space model and demonstrate how to obtain it. We are careful to make a clear distinction between ES methods and their state-space representation called ES models. Lastly, Section 6.3 is dedicated to training of state-space models before forecasts can be made. The experimental results for ES models is presented in Section 12.1.

Forecasting with ES models also require various parameters and initial values to be estimated. For clarity we list them all here and return to their estimation in Section 6.3. The aforementioned parameters are $\alpha, \beta, \beta^*, \gamma, \phi$, and initial values are $l_0, b_0, s_0, s_{-1}, \ldots, s_{-m+1}$.

6.1 ES methods

All ES methods in this chapter can be described using the component form. Component form in general consists of a forecast equation and a number of smoothing equations. Smoothing equations are designed to model the three components of decomposed time series: level, trend and seasonality.

We divide ES methods into three categories called single ES methods, double ES methods and triple ES methods according to the number of components (level, trend, seasonality) they include in their equations.

6.1.1 Single ES

Single ES, also known as simple ES or first order ES, was first described by Brown [49]. It is best suited for data that does not exhibit any trends or seasonal patterns. In its simplest form ES forecasts the next observation $\hat{y}_{t+1|t}$ from previous observations $\boldsymbol{y}_t$ as

$$\hat{y}_{t+1|t} = \alpha y_t + \alpha(1-\alpha)y_{t-1} + \alpha(1-\alpha)^2 y_{t-2} + \cdots + \alpha(1-\alpha)^{t-1}y_1.$$

After realizing that $\hat{y}_{t+1} - (1-\alpha)\hat{y}_{t|t-1} = \alpha y_t$ we arrive at the so-called *component form* for single ES:

$$\hat{y}_{t+h|t} = l_t \tag{6.1}$$
$$l_t = \alpha y_t + (1-\alpha)l_{t-1} \tag{6.2}$$

where $0 \leq \alpha \leq 1$ is the smoothing parameter and $\boldsymbol{l}$ is the level component. l_t is in essence the weighted average of the most recent observation y_t and the previous one step ahead in sample forecast $\hat{y}_{t|t-1}$. Equation (6.1) is the *forecast equation* and Equation (6.2) is the *smoothing equation* for level component or simply level equation. It is important to note that for forecast horizons with $h > 1$ this definition results in the forecast function being "flat" as $\hat{y}_{t+h|t} = \hat{y}_{t+1|t} = l_t$. This is because single ES works best when there is no trend or seasonality present in the series and therefore should not be used as a stand-alone technique, but more as a foundation for more general ES methods that can cope with trend and seasonality. For this reason it is also not suitable for forecasting energy consumption data, as we expect it to exhibit seasonality and also sometimes trend.

If follows that large α leads to more weight being assigned to more recent observations and vice-versa. The influence of l_0 on the outcome is usually negligible as the corresponding weight it small. However, short time series or very small α necessitate also l_0 being chosen carefully. Values of both α and l_0 can be hand-picked or estimated objectively in a training process described in Section 6.3.

6.1.2 Double ES

Double ES is in some sources also referred to as second order ES. It encompasses all methods employing two components in their respective component forms. Without any loss of generality, in this section we present methods using level and trend components. How to incorporate seasonal component into ES equations is described later in Section 6.1.3.

We divide double ES methods by the relationship between their level and trend components into four subcategories.

Additive trend

The first double ES method was introduced by Holt [50] when he expanded upon single ES to forecast time series exhibiting trend. He did this by adding a trend component $\boldsymbol{b}$ to component form, which is now known as Holt's linear trend method:

$$\begin{aligned} \hat{y}_{t+h|t} &= l_t + hb_t \\ l_t &= \alpha y_t + (1-\alpha)(l_{t-1} + b_{t-1}) \\ b_t &= \beta^*(l_t - l_{t-1}) + (1-\beta^*)b_{t-1} \end{aligned}$$

where $0 \leq \alpha, \beta^* \leq 1$ are smoothing parameters, $\boldsymbol{l}$ is the level component and $\boldsymbol{b}$ is the trend component. Here the forecast function is no longer flat, but linear with respect to the length of forecast horizon h.

As in the case of single ES, l_t is the weighted average of the most recent observation y_t and the previous one step ahead in sample forecast $\hat{y}_{t|t-1}$. On the other hand, b_t from trend equation is the weighted average of the most recent trend $l_t - l_{t-1}$ and previous estimation of trend b_{t-1}.

Multiplicative trend

Another approach to double ES is to allow the relationship between level $\boldsymbol{l}$ and trend $\boldsymbol{b}$ to be multiplicative rather than additive:

$$\begin{aligned}\hat{y}_{t+h|t} &= l_t b_t^h \\ l_t &= \alpha y_t + (1-\alpha)(l_{t-1}b_{t-1}) \\ b_t &= \beta^* \frac{l_t}{l_{t-1}} + (1-\beta^*)b_{t-1}\end{aligned}$$

where α, β^* are smoothing parameters. In this method b_t can be interpreted as a growth rate. The forecast function here is exponential with respect to h as opposed to linear.

Damped additive trend

Empirical evidence suggests that both linear and exponential methods tent to over-forecast when it comes to longer forecast horizons. To mitigate this phenomenon Gardner and McKenzie [51] introduced a new parameter $0 < \phi < 1$ to the linear trend method:

$$\begin{aligned}\hat{y}_{t+h|t} &= l_t + \phi_h b_t \\ l_t &= \alpha y_t + (1-\alpha)(l_{t-1} + \phi b_{t-1}) \\ b_t &= \beta^*(l_t - l_{t-1}) + (1-\beta^*)\phi b_{t-1}\end{aligned}$$

where $0 \leq \alpha, \beta^* \leq 1$ are smoothing parameters, $\boldsymbol{l}$ is the level and $\boldsymbol{b}$ is the trend component and $\phi_h = (\phi + \phi^2 + \cdots + \phi^h)$.

While shorter forecast horizons h still produce trended forecasts, one can prove [48] that longer horizons h result in forecasts approaching a constant, i.e. with fixed t the following holds:

$$\lim_{h\to\infty} \hat{y}_{t+h|t} = l_t + \phi \frac{b_t}{1-\phi}$$

Damped multiplicative trend

Similar approach was adapted by Taylor [52] while modifying double ES with multiplicative trend to include damping parameter $0 < \phi < 1$:

$$\begin{aligned}\hat{y}_{t+h|t} &= l_t b_t^{\phi_h} \\ l_t &= \alpha y_t + (1-\alpha)(l_{t-1}b_{t-1}^{\phi}) \\ b_t &= \beta^* \frac{l_t}{l_{t-1}} + (1-\beta^*)b_{t-1}^{\phi}\end{aligned}$$

where $0 \leq \alpha, \beta^* \leq 1$ are smoothing parameters, $\boldsymbol{l}$ is the level component, $\boldsymbol{b}$ is the trend component and $\phi_h = (\phi + \phi^2 + \cdots + \phi^h)$.

6.1.3 Triple ES

As the name suggests, triple or third order ES takes into account three components: level, trend and seasonality. In literature it is also refereed to as Holt-Winters method. In the same paper where he intruduced double ES [50], Holt also described how to exploit the seasonality of time series if one is present. Independently on Holt, a similar approach was presented also by Winters [53].

Although triple ES is able to capture the intrinsic seasonality of the energy consumption data to some degree, it can only do so for one type of seasonality. As the data usually contains multiple seasonalities (daily, weekly and annual), either another type of forecasting technique or some sort of preprocessing must be considered in order for ES to be efficient. The preprocessing techniques considered for ES are described later in Section 10.3.

Additive seasonality

This method is useful when the nature of seasonality present in the time series is additive, that is seasonal changes are not dependant on the level of the series, they depend only on time.

To construct the component form we will extend double ES with linear trend, but any of the previous methods can be modified analogically:

$$\begin{aligned}
\hat{y}_{t+h|t} &= l_t + hb_t + s_{t-m+h} \\
l_t &= \alpha(y_t - s_{t-m}) + (1-\alpha)(l_{t-1} + b_{t-1}) \\
b_t &= \beta^*(l_t - l_{t-1}) + (1-\beta^*)b_{t-1} \\
s_t &= \gamma(y_t - l_{t-1} - b_{t-1}) + (1-\gamma)s_{t-m}
\end{aligned}$$

where $0 \leq \alpha, \beta^*, \gamma \leq 1$ are smoothing parameters, $\boldsymbol{l}$ is the level component, $\boldsymbol{b}$ is the trend component, $\boldsymbol{s}$ is the seasonality component and m is the frequency of the time series $\boldsymbol{y}$.

Level component l_t is now a weighted average of the last observation adjusted for seasonality $(y_t - s_{t-m})$ and the nonseasonal forecast $(l_{t-1} + b_{t-1})$. Trend component b_t is the same as in linear double ES, i.e. the weighted average of the most recent trend $l_t - l_{t-1}$ and previous estimation of trend b_{t-1}, and seasonal component s_t is a weighted average of last observation adjusted for level and trend $y_t - l_{t-1} - b_{t-1}$ and seasonal component exactly one season ago s_{t-m}.

Multiplicative seasonality

For time series whose seasonal component is not only dependant on time but is also proportional to the level of the series, a different approach is more suitable.

It is again possible to extend any of the double ES methods. For example component form of double ES with damped multiplicative trend and multiplicative seasonality looks like this:

$$\begin{aligned}\hat{y}_{t+h|t} &= l_t b_t^{\phi_h} s_{t-m+h}\\ l_t &= \alpha(\frac{y_t}{s_{t-m}}) + (1-\alpha) l_{t-1} b_{t-1}^{\phi}\\ b_t &= \beta^*(\frac{l_t}{l_{t-1}}) + (1-\beta^*) b_{t-1}^{\phi}\\ s_t &= \gamma(\frac{y_t}{l_{t-1} b_{t-1}^{\phi}}) + (1-\gamma) s_{t-m}\end{aligned}$$

where $0 \leq \alpha, \beta^*, \gamma \leq 1$ are the smoothing parameters, $0 \leq \phi \leq 1$ is the damping parameter, $\boldsymbol{l}$ is the level component, $\boldsymbol{b}$ is the trend component, $\boldsymbol{s}$ is the seasonality component and $\phi_h = (\phi + \phi^2 + \cdots + \phi^h)$.

6.2 ES models

In this section we demonstrate that under every ES method lies a state space model. This enables us to optimise smoothing parameters and initial values of components for a particular method and also compare ES methods with each other in an objective manner. For each ES method we will consider two state space models: one with additive and the other with multiplicative errors.

To demonstrate the process of converting an ES method into the underlying state space model, we will use double ES with additive trends. However, this procedure can be altered to suit any of the previous state space models.

6.2.1 Additive errors

Let us consider the component form of double ES with additive trend:

$$\begin{aligned}\hat{y}_{t+h|t} &= l_t + h b_t && (6.3)\\ l_t &= \alpha y_t + (1-\alpha)(l_{t-1} + b_{t-1}) && (6.4)\\ b_t &= \beta^*(l_t - l_{t-1}) + (1-\beta^*) b_{t-1}. && (6.5)\end{aligned}$$

From forecast equation (6.3) it follows that $\hat{y}_{t|t-1} = l_{t-1} + b_{t-1}$. Using this formula one can rewrite level equation (6.4) as

$$l_t = \alpha y_t + (1-\alpha)\hat{y}_{t|t-1}.$$

In this form l_t can be thought of as the weighted average between the most recent observation y_t and the previous one step ahead in sample forecast $\hat{y}_{t|t-1}$. By rearranging this representation of level equation even further we get

$$\begin{aligned} l_t &= \alpha y_t + (1-\alpha)\hat{y}_{t|t-1}\\ &= \hat{y}_{t|t-1} + \alpha(y_t - \hat{y}_{t|t-1}).\end{aligned}$$

After we let $\epsilon_t = y_t - \hat{y}_{t|t-1}$ we obtain the *error correction form* of level equation:

$$l_t = (l_{t-1} + b_{t-1}) + \alpha\epsilon_t \qquad (6.6)$$

In this form ϵ_t is one-step-ahead forecast error, or residual, that corrects the estimated level. We can rearrange trend equation (6.5) in a similar manner:

$$\begin{aligned}
b_t &= \beta^*(l_t - l_{t-1}) + (1-\beta^*)b_{t-1} \\
&= b_{t-1} + \beta^*(l_t - (l_{t-1} + b_{t-1})) \\
&= b_{t-1} + \beta^*((l_{t-1} + b_{t-1}) + \alpha\epsilon_t - (l_{t-1} + b_{t-1})) \\
&= b_{t-1} + \alpha\beta^*\epsilon_t.
\end{aligned}$$

For convenience we let $\beta = \alpha\beta^*$ to produce error correction form of trend equation:

$$b_t = b_{t-1} + \beta\epsilon_t. \tag{6.7}$$

Furthermore, from $\epsilon_t = (y_t - \hat{y}_{t|t-1}) = y_t - (l_{t-1} + b_{t-1})$ it follows that

$$y_t = (l_{t-1} + b_{t-1}) + \epsilon_t \tag{6.8}$$

If we combine the error correction forms of level (6.6) and trend (6.7) equations with Equation (6.8), we obtain the spate-space model underlying the double ES with additive trend. This can be demonstrated by defining state vector $\boldsymbol{x}_t = (l_t, b_t)$ and then transforming (6.8),(6.6) and (6.7) into the standard state-space notation consisting of observation and state equations:

$$\begin{aligned}
y_t &= \begin{pmatrix} 1 \\ 1 \end{pmatrix}' \boldsymbol{x}_{t-1} + \epsilon_t \\
\boldsymbol{x}_t &= \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \boldsymbol{x}_{t-1} + \begin{pmatrix} \alpha \\ \beta \end{pmatrix} \epsilon_t.
\end{aligned}$$

In this model $\boldsymbol{\epsilon}$ is modelled by the residuals and thus the state-space model is fully specified.

6.2.2 Multiplicative errors

To derive error correction form with multiplicative errors for double ES with additive trend, we again start with rearranging the level equation (6.4) in the following manner:

$$\begin{aligned}
l_t &= \alpha y_t + (1-\alpha)(l_{t-1} + b_{t-1}) \\
&= \alpha y_t + (1-\alpha)\hat{y}_{t|t-1} \\
&= \hat{y}_{t|t-1} + \alpha(y_t - \hat{y}_{t|t-1}) \\
&= \hat{y}_{t|t-1} + \alpha\hat{y}_{t|t-1}\frac{y_t - \hat{y}_{t|t-1}}{\hat{y}_{t|t-1}} \\
&= \hat{y}_{t|t-1}(1 + \alpha\frac{y_t - \hat{y}_{t|t-1}}{\hat{y}_{t|t-1}}) \\
&= (l_{t-1} + b_{t-1})(1 + \alpha\epsilon_t)
\end{aligned}$$

where $\epsilon_t = (y_t - \hat{y}_{t|t-1})/\hat{y}_{t|t-1}$ is now the most recent relative one-step forecast error. From here one can follow the same steps as before to get to the underlying state-space model:

$$\begin{aligned}
y_t &= (l_{t-1} + b_{t-1})(1 + \epsilon_t) \\
l_t &= (l_{t-1} + b_{t-1})(1 + \alpha\epsilon_t) \\
b_t &= b_{t-1} + \beta(l_{t-1} + b_{t-1})\epsilon_t
\end{aligned}$$

or equivalently:

$$\begin{aligned} y_t &= \begin{pmatrix}1\\1\end{pmatrix}' \boldsymbol{x}_{t-1} + \begin{pmatrix}1\\1\end{pmatrix}' \boldsymbol{x}_{t-1}\epsilon_t \\ \boldsymbol{x}_t &= \begin{pmatrix}1 & 1\\0 & 1\end{pmatrix} \boldsymbol{x}_{t-1} + \begin{pmatrix}1\\1\end{pmatrix}' \boldsymbol{x}_{t-1} \begin{pmatrix}\alpha\\\beta\end{pmatrix} \epsilon_t \end{aligned} \tag{6.9}$$

with white noise process $\boldsymbol{\epsilon}$ modelled by residuals.

Notice that this is a nonlinear innovations state-space model, which usually means that it is difficult to handle while estimating and forecasting. However, innovations form of state-space models enables us to compute forecasts and likelihood function with no additional effort compared to the additive error model (see Section 6.3).

6.2.3 General state-space representation

The list of ES methods discussed so far is not exhaustive. By combining trend types (none, additive, multiplicative, damped additive, damped multiplicative) with variations in seasonality (none, additive, multiplicative) we can define fifteen distinct methods, each of which has two underlying state-space models: one with additive and one with multiplicative errors. Thus the total number of possible models is thirty.

As we demonstrated in Sections 6.2.1 and 6.2.2, an innovations state-space model lies underneath every ES method. If we define state vector as $\boldsymbol{x}_t = (l_t, b_t, s_t, s_{t-1}, \dots, s_{t-m+1})$, the observation and state equation for the general representation is of the form:

$$y_t = w(\boldsymbol{x}_{t-1}) + r(\boldsymbol{x}_{t-1})\epsilon_t \tag{6.10}$$

$$\boldsymbol{x}_t = \boldsymbol{f}(\boldsymbol{x}_{t-1}) + \boldsymbol{g}(\boldsymbol{x}_{t-1})\epsilon_t \tag{6.11}$$

where $\boldsymbol{\epsilon}$ are modelled by the residuals. For a model with additive errors we let $r(\boldsymbol{x}_{t-1}) = 1$ to obtain $y_t = w(\boldsymbol{x}_{t-1}) + \epsilon_t$. On the other hand $r(\boldsymbol{x}_{t-1}) = w(\boldsymbol{x}_{t-1})$ results in a model with multiplicative errors where $y_t = w(\boldsymbol{x}_{t-1})(1 + \epsilon_t)$.

Each of the thirty ES methods can be represented in the form of (6.10) and (6.11). The ones with additive errors are presented in Table 6.1, while Table 6.2 contains those with multiplicative errors. To simplify the notation we use $\beta = \alpha\beta^*$.

Numerical difficulties may arise when a combination of trend, seasonality and error requires a division by a state component which may at some point result in a division by zero. This issue pertains to models with additive errors and either multiplicative trend or multiplicative seasonality and also includes models with multiplicative errors, multiplicative trend and additive seasonality. Another important fact about models with multiplicative error is that they are not numerically stable when time series is not strictly positive, i.e. the data contains zeros or negative values.

We can obtain point forecasts $\hat{\boldsymbol{y}}_{t+h|t}$ from ES models with the procedure described in Section 5.5. These forecasts are identical to the point forecasts generated by the corresponding ES method.

Trend	Seasonality N	Seasonality A	Seasonality M
N	$y_t = l_{t-1} + \epsilon_t$ $l_t = l_{t-1} + \alpha\epsilon_t$	$y_t = l_{t-1} + s_{t-m} + \epsilon_t$ $l_t = l_{t-1} + \alpha\epsilon_t$ $s_t = s_{t-m} + \gamma\epsilon_t$	$y_t = l_{t-1}s_{t-m} + \epsilon_t$ $l_t = l_{t-1} + \alpha\epsilon_t/s_{t-m}$ $s_t = s_{t-m} + \gamma\epsilon_t/l_{t-1}$
A	$y_t = l_{t-1} + b_{t-1} + \epsilon_t$ $l_t = l_{t-1} + b_{t-1} + \alpha\epsilon_t$ $b_t = b_{t-1} + \beta\epsilon_t$	$y_t = l_{t-1} + b_{t-1} + s_{t-m} + \epsilon_t$ $l_t = l_{t-1} + b_{t-1} + \alpha\epsilon_t$ $b_t = b_{t-1} + \beta\epsilon_t$ $s_t = s_{t-m} + \gamma\epsilon_t$	$y_t = (l_{t-1} + b_{t-1})s_{t-m} + \epsilon_t$ $l_t = l_{t-1} + b_{t-1} + \alpha\epsilon_t/s_{t-m}$ $b_t = b_{t-1} + \beta\epsilon_t/s_{t-m}$ $s_t = s_{t-m} + \gamma\epsilon_t/(l_{t-1} + b_{t-1})$
A_d	$y_t = l_{t-1} + \phi b_{t-1} + \epsilon_t$ $l_t = l_{t-1} + \phi b_{t-1} + \alpha\epsilon_t$ $b_t = \phi b_{t-1} + \beta\epsilon_t$	$y_t = l_{t-1} + \phi b_{t-1} + s_{t-m} + \epsilon_t$ $l_t = l_{t-1} + \phi b_{t-1} + \alpha\epsilon_t$ $b_t = \phi b_{t-1} + \beta\epsilon_t$ $s_t = s_{t-m} + \gamma\epsilon_t$	$y_t = (l_{t-1} + \phi b_{t-1})s_{t-m} + \epsilon_t$ $l_t = l_{t-1} + \phi b_{t-1} + \alpha\epsilon_t/s_{t-m}$ $b_t = \phi b_{t-1} + \beta\epsilon/s_{t-m}$ $s_t = s_{t-m} + \gamma\epsilon_t/(l_{t-1} + \phi b_{t-1})$
M	$y_t = l_{t-1}b_{t-1} + \epsilon_t$ $l_t = l_{t-1}b_{t-1} + \alpha\epsilon_t$ $b_t = b_{t-1} + \beta\epsilon_t/l_{t-1}$	$y_t = l_{t-1}b_{t-1} + s_{t-m} + \epsilon_t$ $l_t = l_{t-1}b_{t-1} + \alpha\epsilon_t$ $b_t = b_{t-1} + \beta\epsilon_t/l_{t-1}$ $s_t = s_{t-m} + \gamma\epsilon_t$	$y_t = l_{t-1}b_{t-1}s_{t-m} + \epsilon_t$ $l_t = l_{t-1}b_{t-1} + \alpha\epsilon_t/s_{t-m}$ $b_t = b_{t-1} + \beta\epsilon_t/(s_{t-m}l_{t-1})$ $s_t = s_{t-m} + \gamma\epsilon_t/(l_{t-1}b_{t-1})$
M_d	$y_t = l_{t-1}b_{t-1}^{\phi} + \epsilon_t$ $l_t = l_{t-1}b_{t-1}^{\phi} + \alpha\epsilon_t$ $b_t = b_{t-1}^{\phi} + \beta\epsilon_t/l_{t-1}$	$y_t = l_{t-1}b_{t-1}^{\phi} + s_{t-m} + \epsilon_t$ $l_t = l_{t-1}b_{t-1}^{\phi} + \alpha\epsilon_t$ $b_t = b_{t-1}^{\phi} + \beta\epsilon_t/l_{t-1}$ $s_t = s_{t-m} + \gamma\epsilon_t$	$y_t = l_{t-1}b_{t-1}^{\phi}s_{t-m} + \epsilon_t$ $l_t = l_{t-1}b_{t-1}^{\phi} + \alpha\epsilon_t/s_{t-m}$ $b_t = b_{t-1}^{\phi} + \beta\epsilon_t/(s_{t-m}l_{t-1})$ $s_t = s_{t-m} + \gamma\epsilon_t/(l_{t-1}b_{t-1}^{\phi})$

Table 6.1: ES models with additive errors [11]

Trend	Seasonality		
	N	A	M
N	$y_t = l_{t-1}(1+\epsilon_t)$ $l_t = l_{t-1}(1+\alpha\epsilon_t)$	$y_t = (l_{t-1}+s_{t-m})(1+\epsilon_t)$ $l_t = l_{t-1}+\alpha(l_{t-1}+s_{t-m})\epsilon_t$ $s_t = s_{t-m}+\gamma(l_{t-1}+s_{t-m})\epsilon_t$	$y_t = l_{t-1}s_{t-m}(1+\epsilon_t)$ $l_t = l_{t-1}(1+\alpha\epsilon_t)$ $s_t = s_{t-m}(1+\gamma\epsilon_t)$
A	$y_t = (l_{t-1}+b_{t-1})(1+\epsilon_t)$ $l_t = (l_{t-1}+b_{t-1})(1+\alpha\epsilon_t)$ $b_t = b_{t-1}+\beta(l_{t-1}+b_{t-1})\epsilon_t$	$y_t = (l_{t-1}+b_{t-1}+s_{t-m})(1+\epsilon_t)$ $l_t = l_{t-1}+b_{t-1}+\alpha(l_{t-1}+b_{t-1}+s_{t-m})\epsilon_t$ $b_t = b_{t-1}+\beta(l_{t-1}+b_{t-1}+s_{t-m})\epsilon_t$ $s_t = s_{t-m}+\gamma(l_{t-1}+b_{t-1}+s_{t-m})\epsilon_t$	$y_t = (l_{t-1}+b_{t-1})s_{t-m}(1+\epsilon_t)$ $l_t = (l_{t-1}+b_{t-1})(1+\alpha\epsilon_t)$ $b_t = b_{t-1}+\beta(l_{t-1}+b_{t-1})\epsilon_t$ $s_t = s_{t-m}(1+\gamma\epsilon_t)$
A_d	$y_t = (l_{t-1}+\phi b_{t-1})(1+\epsilon_t)$ $l_t = (l_{t-1}+\phi b_{t-1})(1+\alpha\epsilon_t)$ $b_t = \phi b_{t-1}+\beta(l_{t-1}+\phi b_{t-1})\epsilon_t$	$y_t = (l_{t-1}+\phi b_{t-1}+s_{t-m})(1+\epsilon_t)$ $l_t = l_{t-1}+\phi b_{t-1}+\alpha(l_{t-1}+\phi b_{t-1}+s_{t-m})\epsilon_t$ $b_t = \phi b_{t-1}+\beta(l_{t-1}+\phi b_{t-1}+s_{t-m})\epsilon_t$ $s_t = s_{t-m}+\gamma(l_{t-1}+\phi b_{t-1}+s_{t-m})\epsilon_t$	$y_t = (l_{t-1}+\phi b_{t-1})s_{t-m}(1+\epsilon_t)$ $l_t = (l_{t-1}+\phi b_{t-1})(1+\alpha\epsilon_t)$ $b_t = \phi b_{t-1}+\beta(l_{t-1}+\phi b_{t-1})\epsilon_t$ $s_t = s_{t-m}(1+\gamma\epsilon_t)$
M	$y_t = l_{t-1}b_{t-1}(1+\epsilon_t)$ $l_t = l_{t-1}b_{t-1}(1+\alpha\epsilon_t)$ $b_t = b_{t-1}(1+\beta\epsilon_t)$	$y_t = (l_{t-1}b_{t-1}+s_{t-m})(1+\epsilon_t)$ $l_t = l_{t-1}b_{t-1}+\alpha(l_{t-1}b_{t-1}+s_{t-m})\epsilon_t$ $b_t = b_{t-1}+\beta(l_{t-1}b_{t-1}+s_{t-m})\epsilon_t/l_{t-1}$ $s_t = s_{t-m}+\gamma(l_{t-1}b_{t-1}+s_{t-m})\epsilon_t$	$y_t = l_{t-1}b_{t-1}s_{t-m}(1+\epsilon_t)$ $l_t = l_{t-1}b_{t-1}(1+\alpha\epsilon_t)$ $b_t = b_{t-1}(1+\beta\epsilon_t)$ $s_t = s_{t-m}(1+\gamma\epsilon_t)$
M_d	$y_t = l_{t-1}b_{t-1}^{\phi}(1+\epsilon_t)$ $l_t = l_{t-1}b_{t-1}^{\phi}(1+\alpha\epsilon_t)$ $b_t = b_{t-1}^{\phi}(1+\beta\epsilon_t)$	$y_t = (l_{t-1}b_{t-1}^{\phi}+s_{t-m})(1+\epsilon_t)$ $l_t = l_{t-1}b_{t-1}^{\phi}+\alpha(l_{t-1}b_{t-1}^{\phi}+s_{t-m})\epsilon_t$ $b_t = b_{t-1}^{\phi}+\beta(l_{t-1}b_{t-1}^{\phi}+s_{t-m})\epsilon_t/l_{t-1}$ $s_t = s_{t-m}+\gamma(l_{t-1}b_{t-1}^{\phi}+s_{t-m})\epsilon_t$	$y_t = l_{t-1}b_{t-1}^{\phi}s_{t-m}(1+\epsilon_t)$ $l_t = l_{t-1}b_{t-1}^{\phi}(1+\alpha\epsilon_t)$ $b_t = b_{t-1}^{\phi}(1+\beta\epsilon_t)$ $s_t = s_{t-m}(1+\gamma\epsilon_t)$

Table 6.2: ES models with multiplicative errors [11]

6.3 Training

In order to apply one of these state-space models in practice we need to choose the type of model to be used (model selection), initial value of state vector $\boldsymbol{x}_0$ (initialisation) and values of smoothing parameters α, β, γ together with the value of damping parameter ϕ (estimation). We use the term training to refer to these three phases.

Throughout this section let $\boldsymbol{x}_t = (l_t, b_t, s_t, s_{t-1}, \dots, s_{t-m+1})$ be the state vector with initial value $\boldsymbol{x}_0 = (l_0, b_0, s_0, s_{-1}, \dots, s_{-m+1})$ and let $\boldsymbol{\theta} = (\alpha, \beta, \gamma, \phi)$ denote the vector of smoothing and damping parameters.

6.3.1 Initialisation

Contrary to other state-space models, ES models assume that the time series $\boldsymbol{y}$ started with y_1 and that there were no prior observations. That is why $\boldsymbol{x}_0$ can be treated as additional parameters.

The initial value of state vector $\boldsymbol{x}_0$ is usually produced by a heuristic. Models used in this thesis use initialisation procedure suggested by Hyndman et al. in 2008 [11]. It is summarised in Algorithm 1.

Algorithm 1 ES model initialisation

```
 1: $m \leftarrow$ frequency of $\boldsymbol{y}_t$
 2: $\boldsymbol{b}_t \leftarrow \mathbf{MAS}_{2\times m}(\boldsymbol{y}_t)$    ▷ estimate the trend component
 3: $\boldsymbol{y}_t^{det} \leftarrow \begin{cases} \boldsymbol{y}_t - \boldsymbol{b}_t, & \text{if additive trend} \\ \boldsymbol{y}_t / \boldsymbol{b}_t, & \text{if multiplicative trend} \end{cases}$    ▷ detrend time series
 4: for each $i = 1, 2, \dots, m$ do    ▷ compute initial seasonal values
 5:     $s_{-m+i} \leftarrow E(y_i^{det}, y_{m+i}^{det}, y_{2m+i}^{det}, \dots, y_{km+i}^{det}, \dots)$    ▷ i.e. the initial seasonal value for December is the average of Decembers
 6: end for
 7: for each $k = 0, 1, \dots$ do    ▷ for each season
 8:     for each $i = 1, 2, \dots, m$ do    ▷ for each observation in the season
 9:         $y_{km+i}^{des} \leftarrow \begin{cases} y_{km+i} - s_{-m+i}, & \text{if additive seasonality} \\ y_{km+i} / s_{-m+i}, & \text{if multiplicative seasonality} \end{cases}$    ▷ produce $\boldsymbol{y}_t^{des}$
10:     end for
11: end for
12: Fit a linear regression model $\beta_0 + \beta_1 t$ to $\boldsymbol{y}_t^{des}$    ▷ see Section 8.1
13: $l_0 \leftarrow \beta_0$    ▷ initial level is the intercept
14: $b_0 \leftarrow \beta_1$    ▷ initial trend is the slope
```

The value of state vector $\boldsymbol{x}_0$ found by Algorithm 1 is refined further along the smoothing parameters $\boldsymbol{\theta}$ in the estimation phase.

6.3.2 Estimation

Once initialised, the parameters $\boldsymbol{\theta}$ are estimated and the initial value of the state-space vector $\boldsymbol{x}_0$ is modified using the likelihood function $\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{x}_0, \boldsymbol{y}_t)$.

Hyndman et al. [11] proved that if we introduce the following definition:

$$\mathcal{L}^*(\boldsymbol{\theta}, \boldsymbol{x}_0, \boldsymbol{y}_t) = t \log \left(\sum_{i=1}^{t} \frac{\epsilon_i^2}{r^2(\boldsymbol{x}_{i-1})} \right) + 2 \sum_{i=1}^{t} \log |r(\boldsymbol{x}_{i-1})|$$

with $r(\boldsymbol{x}_{t-1})$ from the general state-space representation of ES models (6.10), then the following identity holds:

$$\mathcal{L}^*(\boldsymbol{\theta}, \boldsymbol{x}_0, \boldsymbol{y}_t) = -2 \log \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{x}_0, \boldsymbol{y}_t).$$

This means that we can obtain parameters $\boldsymbol{\theta}$ and $\boldsymbol{x}_0$ by minimizing $\mathcal{L}^*$ in

$$\hat{\boldsymbol{\theta}}, \hat{\boldsymbol{x}}_0 = \underset{\boldsymbol{\theta}, \boldsymbol{x}_0}{\arg\min}\, \mathcal{L}^*(\boldsymbol{\theta}, \boldsymbol{x}_0; \boldsymbol{y}_t).$$

In case of ES models this can be done without the Kalman filter by simply iterating the respective observation and state equations in Table 6.1 or 6.2 to obtain the values of ϵ_t and $r(\boldsymbol{x}_{i-1})$.

6.3.3 Model selection

The aim of model selection is to pick the best ES model for a particular problem from among thirty in an objective manner. This requires some kind of a measure of predictive accuracy. The fact that the formula for likelihood function is known makes it possible to use AICc, the Akaike's Information Criterion [54] corrected for small sample bias. Suppose we estimate ES model to obtain $\hat{\boldsymbol{\theta}}$ and $\hat{\boldsymbol{x}}_0$. Then AICc can be obtained as

$$\text{AICc} = \mathcal{L}^*(\hat{\boldsymbol{\theta}}, \hat{\boldsymbol{x}}_0, \boldsymbol{y}_t) + 2k + \frac{2k(k+1)}{n-(k+1)}$$

where k is the total number of parameters of an ES model, i.e. number of parameters in $\hat{\boldsymbol{\theta}}$ plus number of state variables in $\hat{\boldsymbol{x}}_0$. The best model chosen for forecasting is the one with the lowest AICc (see Section 10.2).

7. Autoregressive moving average

In contrast to ES models that approach time series as a combination of level, trend and seasonal components, autoregressive moving average (ARMA) process attempts to separate time series into deterministic time series and stochastic time series (defined in Section 2.3) and model each separately using linear regression (see Section 8.1).

For the deterministic time series ARMA uses probably the most intuitive of models: regression of y_t against past observations $y_{t-1}, y_{t-2}, \ldots$. This models the autocorrelations in the data. For the stochastic time series ARMA uses regression of y_t against a white noise process ϵ_t. In practice the white noise process is usually modelled by past residuals. The motivation is that when residual $\epsilon_t = y_t - \hat{y}_{t|t-1}$ is large (an unexpected outage occurred), the next observation y_{t+1} and therefore the forecast $\hat{y}_{t+1|t}$ should also be affected (outage still affects the load). The deterministic model is called *autoregressive process* and the stochastic model is referred to as *moving average process*.

The theory presented in this chapter is compiled mainly from two books by Hamilton [4], [10], a book by Box and Jenkins [55] and a book by Hyndman et al. [48]. The chapter is divided into three sections. Section 7.1 introduces a number of ARMA processes of increasing complexity, their strengths and shortcomings. In Section 7.2 we derive the state-space representation of ARMA processes called ARMA models. State-space representation is utilised in Section 7.3, where we train ARMA models before forecasting. The experimental results for models in this chapter are presented in Section 12.2.

In this chapter we also we make a clear distinction between ARMA processes, which are more intuitive to understand, and ARMA models, their state-space representations, which enables us to compute likelihood and other statistical properties and therefore can be estimated in an objective manner.

The main assumption of all ARMA processes is that the time series $\boldsymbol{y}$ undergoing analysis is stationary according to the definition described in Section 2.4. Furthermore, since stationary time series have constant mean $\mu = E(\boldsymbol{y})$, we also assume that $\boldsymbol{y}$ has been mean adjusted beforehand to produce $\boldsymbol{y} \leftarrow \boldsymbol{y} - E(\boldsymbol{y})$. To return the forecast $\hat{y}_{t+i|t}$ back to the original scale we can simply reverse the mean-adjustment by $\hat{y}_{t+i|t} + E(\boldsymbol{y})$.

The equations describing various forecasting methods in this chapter contain a number of parameters, namely the regression orders p, q, P, Q, differencing orders d, D and regression parameters $\theta_1, \theta_2, \ldots, \theta_q$ and $\phi_1, \phi_2, \ldots, \phi_p$. In Sections 7.1 and 7.2 we assume that the specific values of these parameters are known. The process describing how they are acquired is described at the end of the chapter in Section 7.3.

7.1 ARMA processes

This Section describes five different ARMA processes:

- moving average (MA) process,
- autoregressive (AR) process,

- autoregressive moving average (ARMA) process,
- autoregressive integrated moving average (ARIMA) process,
- seasonal autoregressive integrated moving average (SARIMA) process.

The most simple ones are MA process and AR process. We consider these to be a special case of a more general ARMA process that combines both of them. Even the more sophisticated ARIMA process and SARIMA process are in essence and ARMA process with additional data preprocessing build in. For this reason we chose ARMA as the general term to refer to processes and models in this chapter.

7.1.1 MA process

The idea behind MA process is that the observation y_t is not only affected by the error ϵ_t at time t, but also by errors encountered before time t. For example, in load forecasting when the system experienced an unexpected outage at time t and therefore the error $\epsilon_t = y_t - \hat{y}_{t|t-1}$ is large, we might very well expect the outage to affect the system also at time $t+1$. The number of previous errors that the process takes into account is called the *order* of the process. An *MA process of order q*, denoted by MA(q), can be then described by a regression-like equation

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

where $\epsilon_t, \epsilon_{t-1}, \ldots, \epsilon_{t-q}$ are generated by a white noise process $\boldsymbol{\epsilon}$ and $\theta_1, \theta_2, \ldots, \theta_q$ are *MA parameters.*

We can use lag operator L, described in Section 2.6, to rewrite the right hand side of the previous equation into a more compact form

$$y_t = \Theta(\mathrm{L})\epsilon_t \tag{7.1}$$

where $\Theta(\mathrm{L}) = 1 + \sum_{i=1}^{q} \theta_i \mathrm{L}^i$ is called the *MA polynomial.*

MA(q) process produces forecasts $\hat{y}_{t+h|t}$ by iterating (7.1) for $i = 1, 2, \ldots, h$ while ignoring the white noise process at each step by setting $\epsilon_{t+i} = 0$.

The intuition behind MA process is that it serves as a short term memory of time series $\boldsymbol{y}$, because as h approaches q the effects of past residuals diminish and when $h > q$ they completely zero out to produce forecasts $\hat{y}_{t+h|t} = E(y_t) = 0$

As discussed in [56], the residuals of a good forecasting method should resemble a white noise process. MA processes therefore adopt the past residuals to generate the white noise process in (7.1), i.e. $\epsilon_t = y_t - \hat{y}_{t|t-1}$.

7.1.2 AR process

AR process is a special case of linear regression that uses lags $y_{t-1}, y_{t-2}, \ldots$ as predictors of dependant variable y_t. The term *autoregressive* captures the fact that it involves regression of the dependant variable against itself.

The total number of dependant variables in the process is called the *order* of autoregressive process. For example, an *AR of order p*, denoted AR(p) is of the following form:

$$y_t = \phi_1 y_{t-1} + \phi_2 y_{t-2} + \cdots + \phi_p y_{t-p} + \epsilon_t$$

where $\phi_1, , \phi_2, \ldots, \phi_p$ are the *AR parameters* and $\boldsymbol{\epsilon}$ is a white noise process.

If we again adopt the lag operator notation, we may rewrite the previous equation as

$$\Phi(\mathrm{L}) y_t = \epsilon_t \tag{7.2}$$

where $\Phi(\mathrm{L}) = 1 - \sum_{i=1}^{p} \phi_i \, \mathrm{L}^i$ is the *AR polynomial.*

Forecasts of $\hat{y}_{t+h|t}$ are obtained in the same way as in MA process, i.e. iterate (7.2) with $\epsilon_{t+1}, \epsilon_{t+2}, \ldots, \epsilon_{t+h} = 0$. As opposed to MA, AR process does not converge to zero as h tends to p. For this reason AR process serves as the long term memory of time series $\boldsymbol{y}$.

7.1.3 ARMA process

ARMA process is, as the name suggests, a combination of AR and MA process. If was first described by Whittle [57]. The justification for ARMA processes stems from *Wold's decomposition theorem.* It states that for any stationary time series $\boldsymbol{y}$ there exists an infinite moving average representation MA(∞), which can be formally expressed as:

$$y_t = \sum_{i=0}^{\infty} \theta_i \, \mathrm{L}^i \, \epsilon_t + x_t \tag{7.3}$$

where L is the lag operator, θ_i are MA parameters satisfying $\sum_{i=0}^{\infty} \theta_i^2 < \infty$ and θ_0=1, $\boldsymbol{x}$ is a deterministic time series and $\boldsymbol{\epsilon}$ is a white noise process.

MA(∞) in Wold's decomposition depends on an infinite number of parameters θ_i. Since $\sum_{i=1}^{\infty} \theta_i^2 < \infty$, it holds that $\lim_{i \to \infty} \theta_i^2 = 0$ and also $\lim_{i \to \infty} \theta_i = 0$. This means that the parameters in MA(∞) are gradually decaying and we can use MA(q) to approximate MA(∞) with arbitrarily close precision. One can further reduce the number of parameters (i.e. the order of MA process) by choosing an appropriate representation of the deterministic time series $\boldsymbol{x}$. For example, with $\boldsymbol{x}$ constant in time we would need a higher order MA process than if $\boldsymbol{x}$ closely resembles $\boldsymbol{y}$ as there are fewer discrepancies to account for. In the extreme case of $\boldsymbol{x} = \boldsymbol{y}$ an MA(0) process is sufficient.

ARMA processes utilises these observations by incorporating AR(p) process to model the deterministic part $\boldsymbol{x}$ of Wold's decomposition theorem. *ARMA process of orders p, q,* denoted ARMA(p, q), is then a combination of AR(p) and MA(q) and, if derived directly from Wold's decomposition, can be represented by

$$y_t = \epsilon_t + \sum_{i=1}^{q} \theta_i \, \mathrm{L}^i \, \epsilon_t + \sum_{i=1}^{p} \phi_i \, \mathrm{L}^i \, y_t \tag{7.4}$$

For convenience we use lag polynomials to rewrite this into a more concise form commonly present in the majority of literature:

$$\Phi(\mathrm{L}) y_t = \Theta(\mathrm{L}) \epsilon_t \tag{7.5}$$

where $\Phi(\mathrm{L}) = 1 - \sum_{i=1}^{p} \phi_i \, \mathrm{L}^i$ is the AR polynomial and $\Theta(\mathrm{L}) = 1 + \sum_{i=1}^{q} \theta_i \, \mathrm{L}^i$ is the MA polynomial.

Time series $\boldsymbol{y}$ with frequency m that displays autocorrelation (3.1) only at lags that are multiples of m is called *purely seasonal.* Purely seasonal $\boldsymbol{y}$ with frequency m can be modelled by a *purely seasonal ARMA process of orders P, Q*, denoted by ARMA$(P, Q)_m$:

$$\Phi(\mathrm{L}^m)y_t = \Theta(\mathrm{L}^m)\epsilon_t$$

where $\Phi(\mathrm{L}^m) = 1-\sum_{i=1}^{P} \phi_i(\mathrm{L}^m)^i$ is the AR polynomial, $\Theta(\mathrm{L}^m) = 1+\sum_{i=1}^{Q} \theta_i(\mathrm{L}^m)^i$ is the MA polynomial and $\boldsymbol{\epsilon}$ is a white noise process.

Purely seasonal ARMA process is not very realistic. The reason for this is that it in fact represents m identical but separate models for time series $(y_1, y_{1+m}, \dots, y_{1+km}, \dots), \dots, (y_m, y_{m+m}, \dots, y_{m+km}, \dots)$ that are completely decoupled from each other. In practice, time series usually cannot be partitioned in such manner. However, the analysis of time series that exhibit some characteristics of purely seasonal time series (e.g. our energy consumption data, see Figure 3.6 and 3.7) can benefit from combining ARMA with purely seasonal ARMA. In this combination ARMA helps purely seasonal ARMA to model the interactions between the separate time series while purely seasonal ARMA helps ARMA with strong autocorrelations related to the frequency m. This combination is explored further in Section 7.1.5.

Similarly to MA and AR processes, ARMA forecasts $\hat{y}_{t+h|t}$ by iterating (7.5) and letting $\epsilon_{t+1}, \epsilon_{t+2}, \dots, \epsilon_{t+h} = 0$. From MA process ARMA also inherits past residuals as the source of white noise, i.e. $\epsilon_t = y_t - \hat{y}_{t|t-1}$.

The only assumption of Wold's decomposition theorem is that the time series $\boldsymbol{y}$ is stationary. Therefore any stationary time series can be approximated by *ARMA*(p, q) process of sufficiently high orders p, q. However, energy load data are generally not stationary, there is usually daily, weekly and annual seasonality and also an upward trend as the number of appliances in household rises. In the following Sections 7.1.4 and 7.1.5 we discuss how to combat this problem.

7.1.4 ARIMA process

ARIMA process were first described by Box and Jenkins [55]. Similarly to ES methods, they consider time series $\boldsymbol{y}$ to be made up of trend, seasonality and error components. However, instead of modelling these components separately like ES does, their approach is to remove the trend and seasonal components altogether at the very beginning through *differencing* and *seasonal differencing*. The "I" in ARIMA stands for *integrated*, which is the opposite of differencing.

Differencing is an operation on time series designed to remove trend from time series through stabilisation of its mean. Time series $\boldsymbol{y}$ is transformed into a *differenced time series* $\triangle\boldsymbol{y}$ by taking the difference between the last two consecutive observations:

$$\triangle y_t = y_t - y_{t-1} = (1 - \mathrm{L})y_t.$$

This is also called *first order differencing.*

Sometimes first order differencing is not enough to make $\boldsymbol{y}$ stationary. In that case the operation can be repeated to produce *second order differencing*:

$$\triangle^2 y_t = \triangle y_t - \triangle y_{t-1} = y_t - y_{t-1} - (y_{t-1} - y_{t-2}) = y_t - 2y_{t-1} + y_{t-2} = (1 - \mathrm{L})^2 y_t$$

Differencing can be repeated for as long as trend remains present in the time series. Generally, *k-th order differencing* is of the form:

$$\triangle^k y_t = (1 - \mathrm{L})^k y_t$$

ARIMA process of orders p, d, q, denoted ARIMA(p, d, q), is then in essence an ARMA(p, q) process applied on d-th order differenced time series $\triangle^d y_t$:

$$\Phi(\mathrm{L})\triangle^d y_t = \Theta(\mathrm{L})\epsilon_t \tag{7.6}$$

where $\Phi(\mathrm{L}) = 1 - \sum_{i=1}^{p} \phi_i \mathrm{L}^i$ is the AR polynomial, $\Theta(\mathrm{L}) = 1 + \sum_{i=1}^{q} \theta_i \mathrm{L}^i$ is the MA polynomial and $\boldsymbol{\epsilon}$ is a white noise process.

Forecasting with ARIMA takes place in two steps. First the time series $\boldsymbol{y}$ is differenced to produce stationary time series $\boldsymbol{y}^* = \triangle^d \boldsymbol{y}$. ARMA process than produces forecasts $\hat{y}^*_{t+h|t}$ as was discussed in Section 7.1.3. Forecasts are transformed back to the original scale by iterating

$$\hat{y}_{t+i|t} = \hat{y}^*_{t+i|t} + (1 - (1 - L)^d)\hat{y}_{t+i|t}$$

for $i = 1, 2, \ldots, h$ with $\hat{y}_{k|t} = y_k$ whenever $k \leq t$.

ARIMA is well suited for some types of energy consumption data that exhibit an upward trend (household becoming more and more electrified). However, one disadvantage of ARIMA process is that it does have problems handling data with prominent seasonality effects. This is addressed in the following Section 7.1.5.

7.1.5 SARIMA process

Energy consumption data usually feature one or more of daily, weekly and annual seasonalities for which the differencing done with ARIMA process might not be enough. SARIMA (Seasonal ARIMA) is the natural extension of ARIMA that, in addition to the previously discussed differencing, also performs *seasonal differencing* to make the data stationary.

Seasonal differencing is similar to differencing in that it is also an operation performed on time series in order to make it stationary. The key difference is that while differencing removes trend through stabilisation of mean, seasonal differencing removes seasonality by stabilising autocovariance. Instead of taking the difference between the last two consecutive observation, *first order seasonal differencing* produces seasonally differenced time series $\triangle_m \boldsymbol{y}$ from $\boldsymbol{y}$ by taking the difference between an observation and the corresponding observation from previous season:

$$\triangle_m y_t = y_t - y_{t-m} = (1 - \mathrm{L}^m)y_t$$

where m is the frequency of time series $\boldsymbol{y}$. The general form of k-th order seasonal difference is then:

$$\triangle_m^k y_t = (1 - \mathrm{L}^m)^k y_t$$

SARIMA process of orders p, d, q and seasonal orders P, D, Q with frequency m, denoted SARIMA$(p, d, q)(P, D, Q)_m$, can be mathematically expressed as:

$$\Phi(\mathrm{L}^m)\Phi(\mathrm{L})\triangle_m^D \triangle^d y_t = \Theta(\mathrm{L}^m)\Theta(\mathrm{L})\epsilon_t \tag{7.7}$$

where $\Phi(\mathrm{L}^m) = 1 - \sum_{i=1}^{P} \phi_i (\mathrm{L}^m)^i$ and $\Phi(\mathrm{L}) = 1 - \sum_{i=1}^{P} \phi_i \mathrm{L}^i$ are the AR polynomials, $\Theta(\mathrm{L}^m) = 1 + \sum_{i=1}^{Q} \theta_i (\mathrm{L}^m)^i$ and $\Theta(\mathrm{L}) = 1 + \sum_{i=1}^{Q} \theta_i \mathrm{L}^i$ are the MA polynomials and $\boldsymbol{\epsilon}$ is a white noise process.

One way to look at SARIMA$(p, d, q)(P, D, Q)_m$ process is that it is in fact an ARMA$(p + P, q + Q)$ process modelling stationary time series $\triangle_m^D \triangle^d y_t$ with a lot of coefficients ϕ_i and θ_i set to zero. Another perspective of looking at SARIMA$(p,d,q)(P,D,Q)_m$ is that it is an ARIMA(p,d,q) process

$$\Phi(\mathrm{L})\triangle^d y_t = \Theta(\mathrm{L})\xi_t$$

whose errors $\boldsymbol{\xi}$ are modelled by SARIMA$(P,D,Q)_m$ process

$$\Phi(\mathrm{L}^m)\triangle_m^D \xi_t = \Theta(\mathrm{L}^m)\epsilon_t.$$

SARIMA is also very similar to ARIMA when producing forecasts. Forecasting procedure is nearly identical to the one described in 7.1.4. ARMA process forecasts $\hat{y}^*_{t+h|t}$ from the stationary time series $\boldsymbol{y}^* = \triangle_m^D \triangle^d \boldsymbol{y}$ and forecasts are transformed back to the original scale by iterating

$$\hat{y}_{t+i|t} = \hat{y}^*_{t+i|t} + (1-(1-L^m)^D(1-L)^d)\hat{y}_{t+i|t}$$

for $i = 1,2,\ldots,h$ with $\hat{y}_{k|t} = y_k$ whenever $k \leq t$.

On one hand SARIMA can handle data exhibiting both trend and seasonality. On the other hand it can effectively model only one type of seasonality, and energy consumption data usually contain daily, weekly and annuals seasonalities. There is a number of techniques to address this deficiency described in Chapter 8 and Section 10.3.

7.2 ARMA models

Similarly to ES methods, all variations of ARMA process also have an underlying state-space representation. This has a number of important implications. First of all, state-space representation enables us to estimate parameters of a particular model in a sophisticated manner. Second of all, we are able to objectively compare models and select the best one.

Since MA and AR processes are a special case of ARMA and both ARIMA and SARIMA processes are an ARMA process on differenced time series, it is sufficient to define state-space representation only for a general ARMA(p,q) process.

Let $\boldsymbol{y}$ be a stationary time series modelled by ARMA(p,q) process. The general form of the state-space model underlying this process can be then expressed as:

$$\begin{aligned} y_t &= \boldsymbol{w}'\boldsymbol{x}_{t-1} + \epsilon_t \\ \boldsymbol{x}_t &= \boldsymbol{F}\boldsymbol{x}_{t-1} + \boldsymbol{g}\epsilon_t \end{aligned}$$

where $\boldsymbol{\epsilon}$ is a white noise process. The model is complete after we specify $\boldsymbol{w}, \boldsymbol{F}$ and $\boldsymbol{g}$, which is the focus of this section.

There are many equivalent state-space representations of the ARMA(p,q) process. In this section we present a formulation from Tsay [58]. Since from Section 5.5 the forecasts of state-space model are in the form of $\hat{y}_{t+h|t} = E(y_{t+h}|\mathbf{y}_t)$, Tsay defines state vector as $\boldsymbol{x}_t = \hat{\boldsymbol{y}}_{t+k|t} = (\hat{y}_{t+1|t}, \hat{y}_{t+2|t}, \ldots, \hat{y}_{t+k|t})$ where $k = \max(p,q)$. Furthermore, let $\phi_i = 0$ for $i > p$ and $\theta_i = 0$ for $i > q$. We can then rewrite (7.5) as

$$\Phi(\mathrm{L})y_t = \Theta(\mathrm{L})\epsilon_t \tag{7.8}$$

where both polynomials now have degree k, i.e. $\Phi(\mathrm{L}) = 1 - \sum_{i=1}^k \phi_i \mathrm{L}^i$ is the AR polynomial and $\Theta(\mathrm{L}) = 1 + \sum_{i=1}^k \theta_i \mathrm{L}^i$ is the MA polynomial.

Since $\boldsymbol{y}$ is assumed to be stationary, according to [20] we can transform (7.8) into

$$y_t = \frac{\Theta(\mathrm{L})}{\Phi(\mathrm{L})}\epsilon_t = \Psi(\mathrm{L})\epsilon_t = \sum_{i=0}^{\infty} \psi_i \, \mathrm{L}^i \, \epsilon_t \tag{7.9}$$

where $\Psi(\mathrm{L}) = \Theta(\mathrm{L})/\Phi(\mathrm{L})$ is possibly infinite degree lag polynomial. This effectively turns ARMA process into an MA(∞) process. We can expand $\Theta(\mathrm{L}) = \Psi(\mathrm{L})\Phi(\mathrm{L})$ to produce

$$(1 - \theta_1 \, \mathrm{L} - \cdots - \theta_k \, \mathrm{L}^k) = (1 + \phi_1 \, \mathrm{L} + \cdots + \phi_k \, \mathrm{L}^k)(\psi_0 + \psi_1 \, \mathrm{L} + \cdots + \psi_k \, \mathrm{L}^k + \dots).$$

If we define $\psi_0 = 1$, we can also find $\psi_1, \psi_2, \dots, \psi_k$ by equating coefficients of lags L raised to the same power. Specifically, for $0 < i \leq k$ the coefficient θ_i adjacent to L^i satisfies

$$-\theta_i = \psi_i + \phi_1\psi_{i-1} + \cdots + \phi_i\psi_0$$

and from that one can express ψ_i as

$$\psi_i = -\theta_i - \sum_{j=1}^{i} \phi_j \psi_{i-j} \tag{7.10}$$

Now we can combine (7.10) with (7.9) to find the relationship between state vectors $\boldsymbol{x}_t$ and $\boldsymbol{x}_{t+1}$. We begin by expressing y_{t+i} for $0 < i \leq k$ using (7.9) as

$$y_{t+i} = \epsilon_{t+i} + \psi_1\epsilon_{t+i-1} + \cdots + \psi_{i-1}\epsilon_{t+1} + \psi_i\epsilon_t + \dots \tag{7.11}$$

From Section 7.1.1 we know that for MA process to produce forecast $\hat{y}_{t+i|t}$, one iterates (7.9) while letting $\epsilon_{t+1}, \epsilon_{t+2}, \dots, \epsilon_{t+i} = 0$. This enables us to eliminate $\psi_1, \psi_2, \dots, \psi_{i-1}$ from (7.11) and forecast $\hat{y}_{t+i|t}$ as

$$\hat{y}_{t+i|t} = \psi_i\epsilon_t + \psi_{i+1}\epsilon_{t-1} + \psi_{i+2}\epsilon_{t-2} + \dots \tag{7.12}$$

We can follow similar steps when forecasting $\hat{y}_{t+i|t-1}$ with $\epsilon_t, \epsilon_{t+1}, \dots, \epsilon_{t+i} = 0$ and obtain

$$\hat{y}_{t+i|t-1} = \psi_{i+1}\epsilon_{t-1} + \psi_{i+2}\epsilon_{t-2} + \dots \tag{7.13}$$

By combining (7.12) with (7.13) we can derive the recursive relationship between forecasts $\hat{y}_{t+i|t}$ and $\hat{y}_{t+i|t-1}$:

$$\hat{y}_{t+i|t} = \hat{y}_{t+i|t-1} + \psi_i\epsilon_t. \tag{7.14}$$

When we consider the special case of this equation where $i = 0$, using $\psi_0 = 1$ we get a relationship between the current observation $y_t = \hat{y}_{t|t}$ and state variable $\hat{y}_{t|t-1}$:

$$y_t = \hat{y}_{t|t-1} + \epsilon_t. \tag{7.15}$$

At this point we have everything we need to formulate the observation equation by defining:

$$\boldsymbol{w} = (1, 0, \dots, 0)$$

For the state equation let us assume that we know the values of state variables in $\boldsymbol{x}_{t-1}$ and want to use them to obtain the next state vector $\boldsymbol{x}_t$:

$$\begin{aligned} \boldsymbol{x}_{t-1} = \hat{\boldsymbol{y}}_{t+k|t-1} &= (\hat{y}_{t|t-1}, \hat{y}_{t+1|t-1}, \dots, \hat{y}_{t+k-1|t-1}) \\ \boldsymbol{x}_t = \hat{\boldsymbol{y}}_{t+k|t} \quad &= (\hat{y}_{t+1|t}, \hat{y}_{t+2|t}, \dots, \hat{y}_{t+k|t}). \end{aligned}$$

The recursive relationship (7.14) together with (7.10) gives us the values of the first $k-1$ elements $\hat{y}_{t+1|t}, \hat{y}_{t+2|t}, \ldots, \hat{y}_{t+k-1|t}$ in $\boldsymbol{x}_t$ immediately. For the last element $\hat{y}_{t+k|t}$ we utilise the summation form of ARMA process (7.4) and simplify if by letting $\epsilon_{t+1}, \epsilon_{t+2}, \ldots, \epsilon_{t+k} = 0$ in

$$\begin{aligned}
\hat{y}_{t+k|t} &= \epsilon_{t+k} + \sum_{i=1}^{k} \theta_i \epsilon_{t+k-i} + \sum_{i=1}^{k} \phi_i \hat{y}_{t+k-i|t} = \\
&= \sum_{i=1}^{k} \phi_i \hat{y}_{t+k-i|t} + \theta_k \epsilon_t = \\
&= \sum_{i=1}^{k} \phi_i (\hat{y}_{t+k-i|t-1} + \psi_{k-i}\epsilon_t) + \theta_k \epsilon_t = \\
&= \sum_{i=1}^{k} \phi_i \hat{y}_{t+k-i|t-1} + \left(\sum_{i=1}^{k} \phi_i \psi_{k-i} + \theta_k \right) \epsilon_t = \\
&= \sum_{i=1}^{k} \phi_i \hat{y}_{t+k-i|t-1} + \psi_k \epsilon_t
\end{aligned}$$

From here we can proceed to the formulation of state equation by defining:

$$\boldsymbol{F} = \left(\begin{array}{c|cccc} 0 & 1 & 0 & \ldots & 0 \\ \vdots & 0 & 1 & & \vdots \\ & \vdots & & \ddots & \\ 0 & 0 & \ldots & & 1 \\ \hline \phi_k & \phi_{k-1} & \ldots & & \phi_1 \end{array} \right) = \begin{pmatrix} \boldsymbol{0}_{k-1} & \boldsymbol{I}_{k-1} \\ \multicolumn{2}{c}{\widetilde{\boldsymbol{\phi}}'} \end{pmatrix} \qquad \boldsymbol{g} = \begin{pmatrix} \psi_1 \\ \psi_2 \\ \psi_3 \\ \vdots \\ \psi_k \end{pmatrix}$$

where $\widetilde{\boldsymbol{\phi}}$ is a vector of AR parameters ϕ_i in reverse order.

The resulting linear innovations state-space model for ARMA(p, q) process is then expressed by the following observation and state equations:

$$y_t = (1, 0, \ldots, 0)' \boldsymbol{x}_{t-1} + \epsilon_t \tag{7.16}$$

$$\boldsymbol{x}_t = \begin{pmatrix} \boldsymbol{0}_{k-1} & \boldsymbol{I}_{k-1} \\ \multicolumn{2}{c}{\widetilde{\boldsymbol{\phi}}'} \end{pmatrix} \boldsymbol{x}_{t-1} + (\psi_1, \psi_2, \ldots, \psi_k)' \epsilon_t \tag{7.17}$$

where $k = \max(p, q)$ and $\boldsymbol{\epsilon}$ is a white noise process.

Forecasts of ARMA models are obtained the same way as for state-space models (see Section 5.5) and are identical to the ones obtained by ARMA processes, i.e. forecast $\hat{y}_{t+h|t}$ is the result of iterating (7.16) and (7.17) with $\epsilon_{t+1}, \epsilon_{t+2}, \ldots, \epsilon_{t+h} = 0$. As the source of white noise the past residuals $\epsilon_t = y_t - \hat{y}_{t|t-1}$ are used.

7.3 Training

Let us consider stationary time series $\boldsymbol{y}$ with frequency m that we want to model with ARMA(p, q) model and let $k = \max(p, q)$. As was the case with ES models, training consists of three phases: initialisation, estimation and model selection.

In initialisation phase (Section 7.3.1) the starting value of state vector $\boldsymbol{x}_0 = (\hat{y}_{1|0}, \hat{y}_{2|0}, \ldots, \hat{y}_{k|0})$ is set. The second phase (Section 7.3.2) explains the estimation of the parameters denoted by $\boldsymbol{\theta} = (\phi_1, \phi_2, \ldots, \phi_k, \psi_1, \psi_2, \ldots, \psi_k)$ where $\phi_i = 0$ whenever $i > p$. Model selection (Section 7.3.3) for ARMA(p, q) model is synonymous with selecting the orders p, q. This is also the phase where general SARIMA$(p, d, q)(P, D, Q)$ model is reduced to ARMA$(p + P, q + Q)$ model by selecting the appropriate order of both types of differencing d and D.

7.3.1 Initialisation

ARMA models postulate that the process generating $\boldsymbol{y}$ started in infinite past. Furthermore $\boldsymbol{y}$ is adjusted for mean and assumed to be stationary, which means $E(\boldsymbol{y}) = 0$. For this reason the initialisation consist of simply letting $\boldsymbol{x}_0 = \mathbf{0}_k$.

7.3.2 Estimation

As was the case with ES models, the theory behind state-space representation of ARMA processes enable us to estimate their parameters using statistical inference. Maximum likelihood estimation is again the method of choice for ARMA models used in this thesis.

If we assume that the orders of ARMA(p, q) are known. The parameters of this model $\boldsymbol{\theta}$ can be then estimated using the Kalman filter and maximum likelihood estimation as described in Section 5.4.

7.3.3 Model selection

Consider the most general $SARIMA(p, d, q)(P, D, Q)_m$ model where m is the frequency of time series $\boldsymbol{y}$. Model selection aims to find orders p, d, q, P, D, Q for this model that minimises some sort of objective model selection criterion. This takes place in two steps.

In the first step statistical tests are used to determine the minimal orders d, D of differencing for $\boldsymbol{y}$ to be stationary. For the purposes of this thesis the KPSS test [59] is used to determine the minimum d and the OCSB test [60] is used to find the minimum D. After differencing, the SARIMA model is effectively reduced to an ARMA model on the differenced time series, which we can estimate using maximum likelihood estimation discussed previously in Section 7.3.2.

The second phase of model selection consist of computing model selection criterion for a number of combinations of orders (p, q, P, Q). In this thesis we minimise the Akaike's Information Criterion AICc. Given p, q, P, Q we first estimate ARMA$(p + P, q + Q)$ as described in Section 7.3.2 to obtain vector of parameters $\hat{\boldsymbol{\theta}}$. Then the AICc can be computed using

$$\text{AICc} = -2 \ln \mathcal{L}(\hat{\boldsymbol{\theta}}; y_1, y_2, \ldots, y_n) + 2l + \frac{l(l+1)}{n - (l+1)}$$

where $l = 2k$ is the length or dimension of $\hat{\boldsymbol{\theta}}$, i.e. the number of parameters of the model.

What exactly the combinations of orders (p, q, P, Q) under consideration are can be specified by a number of strategies, in this thesis we use hill climbing [61].

Let us define an *order landscape*, a five dimensional space consisting of *location* (the specific combination of orders p, q, P, Q) and elevation (AICc of the model).

We define the *neighbourhood function* on locations (p, q, P, Q) as the set of locations

$$\begin{aligned} N(p, q, P, Q) = \{&(p \pm 1, q, P, Q), (p, q \pm 1, P, Q), (p \pm 1, q \pm 1, P, Q), \\ &(p, q, P \pm 1, Q), (p, q, P, Q \pm 1), (p, q, P \pm 1, Q \pm 1)\} \end{aligned}$$

Hill climbing algorithm traverses through this order landscape by continually moving in the direction of the steepest descent. Which direction to take in the next step from location (p, q, P, Q) is determined by the elevation of the neighbours. This means one needs to initialise, estimate and compute AICc for ARMA models constructed from each of the neighbours in $N(p, q, P, Q)$. The search is ended when none of the neighbours have lower elevation than the current location. Starting location is $(0, 0, 0, 0)$

The main advantage of hill climbing is that is considerably speeds up model selection process, because the search is not exhaustive and not all combinations of orders are considered. Slower model selection may not be an issue when training ARMA models only once. However, it has a much bigger impact when training multiple times (Section 10.2) and for different combinations of data preprocessing (Section 10.3).

The drawback of hill climbing is that the selected model is not necessarily the best one as hill climbing can get stuck in local optima.

The entire model selection process is used when selecting the best model in time series cross-validation (see Section 10.2).

8. Autoregressive moving average with exogenous inputs

The models presented in Chapters 6 and 7 allow only for the inclusion of information from the past observations. However, there may be other information pertinent to modelling the time series in question. For short term forecasting of energy consumption data the weather has probably the most significant impact [12], as the conditions outside the house may alter the behaviour of the occupants (stay at home on rainy weekends) and also energy spent on heating. Information about national and school holidays may also help forecasting. There is also a possibility to use artificially constructed variables to model multiple seasonalities, something that ETS and ARMA models are incapable of.

Regression analysis is a statistical process of constructing mathematical models to explain relationships that may exist between dependent and independent variables [62]. More specifically, regression analysis aims to explain how the values of dependant variables reflect changes in any of the independent variables with the other independent variables being fixed. For the purposes of this thesis it is sufficient to consider regression with only one dependant variable representing energy load. As the term "independent" is rather overloaded in statistics and other mathematical disciplines, we refer to independent variables as *predictors*. In our case all the predictors are in the form of time series.

While regression allows for the inclusion of a wide variety of relevant information in the form of predictors, it forgoes modelling the subtle dynamics of time series which can be handled for example by ARMA models.

Therefore, in this chapter we combine regression with ARMA models to take advantage of their different approaches and cancel out or at least mitigate their respective shortcomings. We refer to this combination first as ARMAX processes and later as ARMAX models (ARMA with eXogenous inputs).

In order to produce forecasts of the dependant variable, first the forecasts of predictors must be obtained. It is important to note that the manner of obtaining forecasts of predictors can introduce unwanted bias to the model. For example, as we were unable to obtain historical weather forecasts and were forced to use real weather data instead, the results may be skewed.

This chapter is divided into four sections. In Section 8.1 we describe the general form of linear regression and also Fourier regression. In Section 8.2 we combine linear regression with ARMA process into ARMAX process. ARMAX process is then subsequently transformed into ARMAX model by deriving its state-space representation in Section 8.3. And finally this state-space representation is used while training the models in Section 8.4. The experimental results for models in this chapter are presented in Section 12.3.

We also differentiate between ARMAX processes and ARMAX models. ARMAX process describes the straightforward combination of regression and ARMA process. ARMAX model is used for ARMAX process in its state-space representation. Also throughout this chapter all the parameters used in various equations describing forecasting methods are assumed to be known. These include the regression orders p, q, P, Q, differencing orders d, D and regression parameters

$\beta_0, \beta_1, \dots, \beta_l$, $\theta_1, \theta_2, \dots, \theta_q$ and $\phi_1, \phi_2, \dots, \phi_p$. The process of obtaining their specific values is described at the end of the chapter in Section 8.4.

8.1 Linear regression

Linear regression is a special case of regression where the dependant variable $\boldsymbol{y}$ is assumed to be a linear combination of predictors (time series) $\boldsymbol{z}^{(1)}, \boldsymbol{z}^{(2)}, \dots, \boldsymbol{z}^{(l)}$. The number of predictors l is problem dependant. In general, linear regression can be expressed as

$$y_t = \beta_0 + \beta_1 z_t^{(1)} + \beta_2 z_t^{(2)} + \dots + \beta_l z_t^{(1)} + \xi_t \tag{8.1}$$

where $\beta_0, \beta_1, \dots, \beta_l$ are model parameters and $\boldsymbol{\xi}$ is a white noise process capturing the deviation of the real data from the modelled relationship. The parameter β_0 is in literature usually referred to as the *intercept*.

If we define vectors $\boldsymbol{\beta} = (\beta_0, \beta_1, \dots, \beta_l)$ and $\boldsymbol{z}_t = (1, z_t^{(1)}, z_t^{(2)}, \dots, z_t^{(l)})$, we can rewrite linear regression (8.1) using a more convenient notation

$$y_t = \boldsymbol{z}_t' \boldsymbol{\beta} + \xi_t. \tag{8.2}$$

When using linear regression to produce point forecasts it is necessary to first obtain the forecasts for predictors. In case of time related predictors like national holidays the forecasts are straightforward. Otherwise they need to be forecast separately, as is the case for weather data. Once forecasts for predictors up to $\hat{\boldsymbol{z}}_{t+h|t} = (1, \hat{z}_{t+h|t}^{(1)}, \hat{z}_{t+h|t}^{(2)}, \dots, \hat{z}_{t+h|t}^{(l)})$ are obtained, forecasts of dependant variable up to $\hat{y}_{t+h|t}$ are produced by iterating (8.2) with $\hat{y}_{t+i|t} = \hat{\boldsymbol{z}}_{t+i|t}' \boldsymbol{\beta}$ and $\xi_{t+i} = 0$ for $i = 1, 2, \dots, h$.

8.1.1 Fourier regression

One of the main disadvantages of ETS and ARMA models was their inability to model multiple seasonalities of energy consumption data. If we were to capture these seasonalities with linear regression and model the remaining deviations with ARMA models, maybe the forecasts would improve. To do this we use Fourier's theorem.

Fourier's theorem or *Fourier's decomposition* [63] states that any (reasonably well-behaved) function can be decomposed into simple trigonometric functions. Mathematically speaking, Fourier proved that function $f(x)$ periodic on the interval $0 \le x \le T$ can be written as

$$f(x) = \alpha_0 + \sum_{i=1}^{\infty} \left(\alpha_i \cos i \frac{2\pi}{T} x + \beta_i \sin i \frac{2\pi}{T} x \right) \tag{8.3}$$

where α_i and β_i are Fourier coefficients. In practise the infinite sum in Equation (8.3) is approximated by a finite $\sum_{i=1}^{k} \left(\alpha_i \cos i \frac{2\pi}{T} x + \beta_i \sin i \frac{2\pi}{T} x \right)$ and the number of summands k is refered to as the *Fourier order*.

We adapt a discrete type of Fourier's decomposition by formulating it as linear regression. Let $\boldsymbol{y}$ be time series containing multiple seasonalities with corresponding frequencies $M = (m_1, m_2, \dots, m_l)$. For each of these frequencies we use a separate Fourier's decomposition, which results in l decompositions with orders $k_1, k_2, \ldots, k_l$. These are combined into a linear regression in the following form:

$$y_t = \alpha_0 + \sum_{m \in M} \sum_{i=1}^{k_m} \left(\alpha_{m,i} \cos i \frac{2\pi}{m} t + \beta_{m,i} \sin i \frac{2\pi}{m} t \right) + \xi_t \tag{8.4}$$

where $\alpha_{m,i}$ and $\beta_{m,i}$ are regression parameters and $\boldsymbol{\xi}$ is a white noise process accounting for deviations from the decomposition. The total number of parameters in Fourier's regression is therefore $1 + 2 \cdot \sum_{m \in M} k_m$.

8.2 ARMAX process

In this thesis linear regression is not used to forecast energy consumption directly as the available weather data is hardly enough to forecast energy profile of the whole next day. Instead we use a combination of linear regression and ARMA.

Let $\boldsymbol{z}_t = (1, z_t^{(1)}, z_t^{(2)}, \ldots, z_t^{(l)})$ be a vector with observations from l separate predictors $\boldsymbol{z}^{(1)}, \boldsymbol{z}^{(2)}, \ldots, \boldsymbol{z}^{(l)}$ at time t. For example $\boldsymbol{z}^{(1)}$ can be predictor representing outside temperature, $\boldsymbol{z}^{(2)}$ can represent outside humidity and $\boldsymbol{z}^{(3)}$ the wind speed. Let us for a moment assume that the dependant time series $\boldsymbol{y}$ and every predictor $\boldsymbol{z}^{(1)}, \boldsymbol{z}^{(2)}, \ldots, \boldsymbol{z}^{(l)}$ is stationary.

ARMAX stands for ARMA with eXogenous (or eXternal) regressors and it is a combination of ARMA and linear regression. In the linear regression the errors $\boldsymbol{\xi}$ are assumed to be a white noise process. *ARMAX process of orders* p, q, denoted ARMAX(p, q), models errors $\boldsymbol{\xi}$ with an *ARMA*(p, q) process instead. Mathematically, this can be expressed as

$$y_t = \boldsymbol{z}_t' \boldsymbol{\beta} + \xi_t \tag{8.5}$$

$$\Phi(\mathrm{L}) \xi_t = \Theta(\mathrm{L}) \epsilon_t \tag{8.6}$$

where $\boldsymbol{\beta} = (\beta_0, \beta_1, \ldots, \beta_l)$ are regression parameters, $\Phi(\mathrm{L}) = 1 - \sum_{i=1}^{p} \phi_i \, \mathrm{L}^i$ and $\Theta(\mathrm{L}) = 1 + \sum_{i=1}^{q} \theta_i \, \mathrm{L}^i$ are the AR and MA polynomials and $\boldsymbol{\epsilon}$ is a white noise process.

If time series $\boldsymbol{y}$ or any predictor from $\boldsymbol{z}^{(1)}, \boldsymbol{z}^{(2)}, \ldots, \boldsymbol{z}^{(l)}$ is nonstationary, differencing described in Sections 7.1.4 or 7.1.5 is performed. In order to preserve the relationship between $\boldsymbol{y}$ and the predictors and ensure interpretability, all of the time series in $\boldsymbol{y}, \boldsymbol{z}^{(1)}, \boldsymbol{z}^{(2)}, \ldots, \boldsymbol{z}^{(l)}$ are differenced to the same degree. The resulting process is referred to as ARIMAX(p, d, q) or SARIMAX$(p, d, q)(P, D, Q)_m$ depending on the types and orders of differencing d, D and the frequency of time series m.

As was the case with linear regression, for forecasts up to $\hat{y}_{t+h|t}$ the forecasts of predictors up to $\hat{\boldsymbol{z}}_{t+h|t} = (1, \hat{z}_{t+h|t}^{(1)}, \hat{z}_{t+h|t}^{(2)}, \ldots, \hat{z}_{t+h|t}^{(l)})$ must be known. Then we can simply iterate (8.5) and (8.6) with $\epsilon_{t+1}, \epsilon_{t+2}, \ldots, \epsilon_{t+h} = 0$. When forecasting with ARIMAX and SARIMAX, forecasts are not produced directly because the differencing needs to be taken into account, which is described in Sections 7.1.4 and 7.1.5.

8.3 ARMAX models

One of the advantages of state-space models is that separate state-space models can be easily combined into one. This enables us to combine state-space representation of linear regression with state-space representation of ARMA to produce ARMAX models.

As was the case with ARIMA and SARIMA, ARIMAX and SARIMAX can be considered an ARMAX with additional preprocessing. For this reason we present state-space representation only for ARMAX process. Consider ARMAX(p, q) process with l predictors described by Equations (8.5) and (8.6).

First we derive the state-space representation for the linear regression part of ARMAX(p, q), i.e. the Equation (8.5). This can be easily put into state-space representation by defining the state vector $\boldsymbol{\beta}_t = \boldsymbol{\beta}$ for all t and writing observation and state equations as

$$\begin{aligned} y_t &= \boldsymbol{z}_t' \boldsymbol{\beta}_{t-1} + \xi_t \\ \boldsymbol{\beta}_t &= \boldsymbol{I}_{l+1} \boldsymbol{\beta}_{t-1}. \end{aligned}$$

Using these steps any linear regression can be put into state-state form. We use the term *linear model* to refer to state-space representation of linear regression. Notice that linear model no longer satisfies time-invariant condition because $\boldsymbol{z}_t$ changes with time.

Now consider the ARMA part of ARMAX(p, q), i.e. the Equation (8.6) and let us assume that the following is its state-space representation:

$$\begin{aligned} \xi_t &= \boldsymbol{w}' \boldsymbol{x}_{t-1} + \epsilon_t \\ \boldsymbol{x}_t &= \boldsymbol{F} \boldsymbol{x}_{t-1} + \boldsymbol{g}' \epsilon_t \end{aligned}$$

The process of obtaining this representation is described in Section 7.2. Since $\boldsymbol{\beta}_t = \boldsymbol{\beta}$ is constant, These two state-space models can be straightforwardly combined into an ARMAX(p, q) model by defining the state vector as

$$\boldsymbol{x}_t^* = \begin{pmatrix} \boldsymbol{\beta} \\ \boldsymbol{x}_t \end{pmatrix}$$

and then the observation and state equations as

$$\begin{aligned} y_t &= \begin{pmatrix} \boldsymbol{z}_t \\ \boldsymbol{w} \end{pmatrix}' \boldsymbol{x}_{t-1}^* \\ \boldsymbol{x}_t^* &= \begin{pmatrix} \boldsymbol{I}_{l+1} & \boldsymbol{0}_{(l+1)\times k} \\ \boldsymbol{0}_{k\times(l+1)} & \boldsymbol{F} \end{pmatrix} \boldsymbol{x}_{t-1}^* + \begin{pmatrix} \boldsymbol{0}_{l+1} \\ \boldsymbol{g} \end{pmatrix} \epsilon_t \end{aligned}$$

where $k = \max(p, q)$.

Forecasts for this model are produced by first obtaining the future values of predictors $\hat{\boldsymbol{z}}_{t+h|t} = (1, \hat{z}_{t+h|t}^{(1)}, \hat{z}_{t+h|t}^{(2)}, \ldots, \hat{z}_{t+h|t}^{(l)})$ and then following procedure described in Section 5.5.

8.4 Training

In this section we describe the process of estimating various parameters that were so far assumed to be known. Specifically, the orders for an ARMA model need to be selected, the parameters of this ARMA model and regression parameters need to be estimated and the state vector needs to be initialised. However, before all of this the type and number of predictors needs to be determined. For now we assume them to be known and focus on the rest. We will return to them in Section 12.3.

Consider the an ARMAX model ARMAX(p, q). Let $\boldsymbol{x}_t^* = (\boldsymbol{\beta}, \boldsymbol{x}_t)$ be its state vector with regression parameters $\boldsymbol{\beta} = (\beta_0, \beta_1, \ldots, \beta_l)$, ARMA state vector $\boldsymbol{x}_t$ ARMA parameters $\boldsymbol{\theta} = (\phi_1, \phi_2, \ldots, \phi_k, \psi_1, \psi_2, \ldots, \psi_k)$ where $k = max(p, q)$ and $\phi_i = 0$ whenever $i > p$. Section 8.4.3 describes model selection, i.e. the selection of orders for p, q. It also reduces a general SARIMAX$(p, d, q)(P, D, Q)_m$ model into a simpler ARMAX$(p + P, q + Q)$ model determining the appropriate differencing orders d, D. In Section 8.4.1 the ARMAX model under consideration is initialised by deriving the value of vector $\boldsymbol{x}_0^* = (\boldsymbol{\beta}, \boldsymbol{x}_0)$ and subsequently estimated in Section 8.4.2 by finding the optimal $\hat{\boldsymbol{\theta}}$ and also modifying the value of $\boldsymbol{\beta}$.

8.4.1 Initialisation

The focus of initialisation is defining the value of state vector $\boldsymbol{x}_0^* = (\boldsymbol{\beta}, \boldsymbol{x}_0)$. We do this by ignoring the ARMA part of ARMAX for the moment and focusing on the linear regression part.

In order to initialise $\boldsymbol{\beta}$, consider the following form of linear regression:

$$\boldsymbol{y}_t = \boldsymbol{Z}\boldsymbol{\beta} + \boldsymbol{\xi}_t \tag{8.7}$$

where

$$\boldsymbol{Z} = \begin{pmatrix} 1 & \boldsymbol{z}_1^{(1)} & \boldsymbol{z}_1^{(2)} & \ldots & \boldsymbol{z}_1^{(l)} \\ 1 & \boldsymbol{z}_2^{(1)} & \boldsymbol{z}_2^{(2)} & \ldots & \boldsymbol{z}_2^{(l)} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & \boldsymbol{z}_t^{(1)} & \boldsymbol{z}_t^{(2)} & \ldots & \boldsymbol{z}_t^{(l)} \end{pmatrix}$$

The initial $\boldsymbol{\beta}$ is estimated from (8.7) using maximum likelihood estimation. According to Pollock [64], for linear model maximum likelihood estimation is equivalent to computing $\boldsymbol{\beta}$ using the following:

$$\boldsymbol{\beta} = (\boldsymbol{Z}'\boldsymbol{Z})^{-1}\boldsymbol{Z}'\boldsymbol{y}$$

This initial $\boldsymbol{\beta}$ is further refined in the estimation phase that follows.

For the initialisation of $\boldsymbol{x}_0$, since linear regression assumes $\boldsymbol{\xi}$ to be a white noise process (zero mean) and ARMA assumes the process started in infinite past, we can let $\boldsymbol{x}_0 = \boldsymbol{0}_k$.

8.4.2 Estimation

State-space representation of ARMAX enables us to estimate the parameters $\boldsymbol{\theta}, \boldsymbol{\beta}$ in an objective manner using statistical inference. For the ARMAX models used in this thesis we use the maximum likelihood estimation.

For a model $ARMAX(p, q)$ with known orders p, q, parameters $\boldsymbol{\theta}, \boldsymbol{\beta}$ can be estimated using the Kalman filter and maximum likelihood estimation. The procedure is described in Section 5.4

8.4.3 Model selection

Model selection process of ARMAX models follows that of the ARMA models explained in Section 7.3.3.

First SARIMAX$(p, d, q)(P, D, Q)$ models is reduced to ARMAX$(p+P, q+Q)$ model by determining the minimal differencing orders d, D to ensure that $\boldsymbol{y}$ and all predictors $\boldsymbol{z}^{(1)}, \boldsymbol{z}^{(2)}, \dots, \boldsymbol{z}^{(l)}$ are stationary. The same orders of differencing are used for all variables to preserve their relationship and ensure interpretability.

Then the the orders p, q, P, Q are selected using hill climbing. For each combination of orders the parameters $\boldsymbol{\beta}$ and $\boldsymbol{\theta}$ are estimated. Let $\hat{\boldsymbol{\beta}}$ and $\hat{\boldsymbol{\theta}}$ be the result of the estimation process, i.e. parameters with the maximum likelihood.

The criterion used in hill climbing is again the Akaike's Information Criterion AICc. For ARMAX it is in the form of

$$\text{AICc} = -2\ln \mathcal{L}(\hat{\boldsymbol{\beta}}, \hat{\boldsymbol{\theta}}; y_1, y_2, \dots, y_n) + 2r + \frac{r(r+1)}{n-(r+1)}$$

where $r = (l+1) + 2k$ is the number of parameters of the model, i.e. the sum of lengths or dimensions of $\hat{\boldsymbol{\beta}}$ and $\hat{\boldsymbol{\theta}}$ respectively.

The entire model selection process is used when selecting the best model in time series cross-validation (see Section 10.2).

9. Artificial neural networks

Artificial neural networks, or simply neural networks, are a type of machine learning algorithm that has been successfully deployed to solve a wide range of problems, ranging from the predictions of heart attacks [14] and stock market prices [15] to credid card fraud detection [16] and self-driving cars [17].

Originating from the work by McCulloch and Pitts [65], a neural network is a mathematical model loosely based on brains of animals and humans. Similarly to how a brain is composed of biological neurons, a neural network is a large collection of interconnected processing units called artificial neurons. Individual neurons are able to receive signals from surrounding neurons, transform it and then relay it to other neurons. In order to control the intensity of passing signals, each connection between the neurons, called *synapse*, is parametrised by a (synaptic) *weight*. In most neural networks the neurons are usually designated as either input, hidden or output neurons. The signal then usually passes from input neurons through hidden neurons until it reaches output neurons.

The purpose of neural networks is to model complex systems described by input/output relationships. The input is usually a description of a system state (patient state, past stock prices, car velocity and its surroundings) and the output is either future state of the system (tomorrow's stock price) or hidden system characteristic (patient healthy/ill) or the best course of actions (self-driving car). Neural networks are designed to capture this relationship in the values of their weights. One of the main reasons behind the popularity of neural networks is their ability to capture various linear and nonlinear relationships [66].

Let vector $\boldsymbol{x}$ be an input and let $\boldsymbol{w}$ be a vector containing all the weights of neurons in the network. We use the vector function $\boldsymbol{f_w}$ parametrised by $\boldsymbol{w}$ to describe the output on the network $\boldsymbol{f_w}(\boldsymbol{x})$ given input $\boldsymbol{x}$.

Weights $\boldsymbol{w}$ of a network are estimated in the process called training. In it the network is gradually exposed to an increasing number of input/output pairs from the available data that guide the estimation of weights $\boldsymbol{w}$. After the training process is complete, the network is ready to be applied in practice.

The theory behind neural networks presented in this chapter comes mainly from books by Nielsen [67] and Mitchell [13]. The chapter is divided into four sections. In Section 9.1 we describe artificial neurons, the building blocks of neural networks. Section 9.2 presents multilayer neural networks. The parameters $\boldsymbol{w}$ of neural networks are estimated in the training process explained in Section 9.3. Lastly, Section 9.4 is dedicated to the application of neural networks on time series data. The experimental results for neural networks are presented in Section 12.4.

The notation in this chapter can be in many instances simplified by the "vectorisation" of scalar function ϕ, i.e. applying ϕ to each element of vector $\boldsymbol{x} = (x_1, x_2, \ldots, x_k)$ to produce vector $(\phi(x_1), \phi(x_2), \ldots, \phi(x_k))$. For this purpose we define the result of *element-wise application of function* ϕ on vector $\boldsymbol{x}$ as vector $\boldsymbol{\phi}[\boldsymbol{x}] = (\phi(x_1), \phi(x_2), \ldots, \phi(x_k))$. This notation also applies to derivations of functions $\boldsymbol{\phi}'[\boldsymbol{x}] = (\phi'(x_1), \phi'(x_2), \ldots, \phi'(x_k))$. Note the difference between the element-wise derivation $\boldsymbol{\phi}'[\boldsymbol{x}]$ and the transposition of vector $\boldsymbol{\phi}[\boldsymbol{x}]'$.

We also take care to make a very clear distinction between *parameters* and *hyperparameters*. Parameters refer to all variables estimated by the network itself from the data in the training process, namely the weights $\boldsymbol{w}$. Hyperparameters describes all variables that need to be set before the training process begin, such as the number of neurons and structure (topology) of the network and other variables modifying the training process such as learning rate λ. The values of hyperparameters are in practice usually chosen using trial and error, grid search or some sort of optimisation algorithm (genetic algorithms). This process is described in more detail in Section 12.4.

9.1 Artificial neuron

Artificial neuron, or simply *neuron*, is an elementary unit of neural networks. In essence it is a mathematical representation of a biological neuron. In a process called *transduction*, biological neurons, such as pain receptors in the skin or photoreceptors in the eye, convert a specific type of stimulus or signal received via their receptors into an *action potential* [68]. Under specific conditions, such as surpassing the pain threshold or sufficient light intensity, this action potential can trigger a chain reaction propagating through other neurons until it reaches the spine or the brain.

In the mathematical model the stimulus or signal is represented by vector $\boldsymbol{x} = (x_1, x_2, \ldots, x_k)$ called the *input vector*. Artificial neuron can be then described as

$$\phi(\boldsymbol{w}'\boldsymbol{x} + b)$$

where *activation function* $\phi : \mathbb{R}^k \to \mathbb{R}$ is mathematical representation of action potential, *weight vector* $\boldsymbol{w} = (w_1, w_2, \ldots, w_k)$ describes the sensitivity of the neuron to various input signals and the *bias* $b \in R$ represents the conditions under which the action potential propagates.

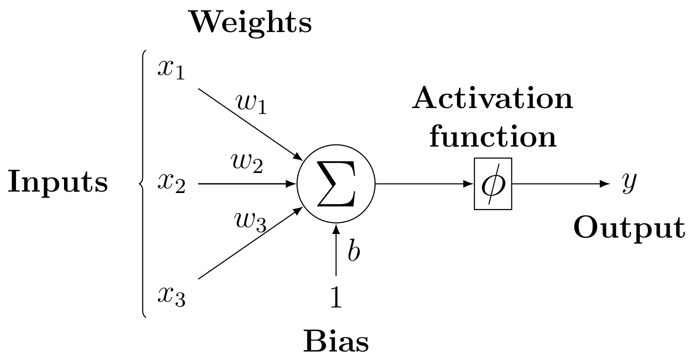

Figure 9.1: Schema of a neuron with three inputs

A graphical representation of an artificial neuron can be found in Figure 9.1. It also motivates a more convenient notation for neurons used for computational purposes. Let us add an additional input $x_0 = 1$ to the input vector $\boldsymbol{x}$. Let also add the associated weight $w_0 = b$ to the weight vector $\boldsymbol{w}$. Artificial neuron then becomes

$$\phi(\boldsymbol{w}'\boldsymbol{x})$$

where $\phi : \mathbb{R}^{k+1} \to \mathbb{R}$ is the activation function, $\boldsymbol{x} = (1, x_1, x_2, \ldots, x_k)$ is the vector of inputs and $\boldsymbol{w} = (b, w_1, w_2, \ldots, w_k)$ is the vector of weights. For convenience let the value $\xi = \boldsymbol{w}'\boldsymbol{x}$ be called the *activation* of neuron. We use the term *output* of neuron to refer to the result of applying the activation function to the activation of neuron, i.e. the value of $\phi(\xi)$.

The exact form of the activation function is problem specific. One of the simplest activation functions is the *binary step activation function*

$$\phi(\xi) = \begin{cases} 0 : & \xi < 0 \\ 1 : & \xi \geq 0. \end{cases}$$

One major disadvantage of binary step activation function is its instability, meaning that a small change in inputs can lead to a big change in the output. A more stable activation function commonly used in neural networks is the *sigmoid activation function*

$$\phi(\xi) = \frac{1}{1 + e^{-\xi}}$$

or the *hyperbolic tangent function* (tanh)

$$\phi(\xi) = \frac{2}{1 + e^{-2\xi}} - 1$$

where e is the Euler's number.

9.2 Multilayer networks

Multilayer network (MLN) is a type of neural network where the neurons are organised into *layers*. Each MLN consists of three types of layers. The first layer is called the *input layer*, the last is referred to as the *output layer* and all other layers in-between them are *hidden layers*. If an MLN contains only one hidden layer, we refer to it as a single-layer network (SLN). A graphical representation of an MLN with two hidden layers is pictured in Figure 9.2.

The input layer contains as many neurons as there are inputs. i-th neuron in input layer is connected only to i-th input x_i with the associated weight $w = 1$, bias $b = 0$ and an identity activation function $\phi(wx_i) = x_i$.

On the other hand, a neuron in a hidden layer is connected to every neuron in the preceding input or hidden layer. The weight vectors of these neurons are estimated in the learning process described in Section 9.3. Usually, all neurons in one layer share the same type of activation function. The type of activation function together with the number of hidden layers and the number of neurons in each layer are hyperparameters that need to be optimised. This can be done using trial and error method or an optimisation algorithm such as genetic algorithms.

The output layer contains as many neurons as there are outputs, one neuron per output. A neuron in the output layer is connected to each neuron in the preceding hidden layer. As in the previous case the weight vectors of neurons in the output layer are estimated in the training process described in Section 9.3. The type of activation function is again shared and a hyperparameter to be optimised.

The motivation behind MLN stems from a paper by Hornik [69], which proves that any continuous function on compact subsets of $\mathbb{R}^n$ can be approximated with

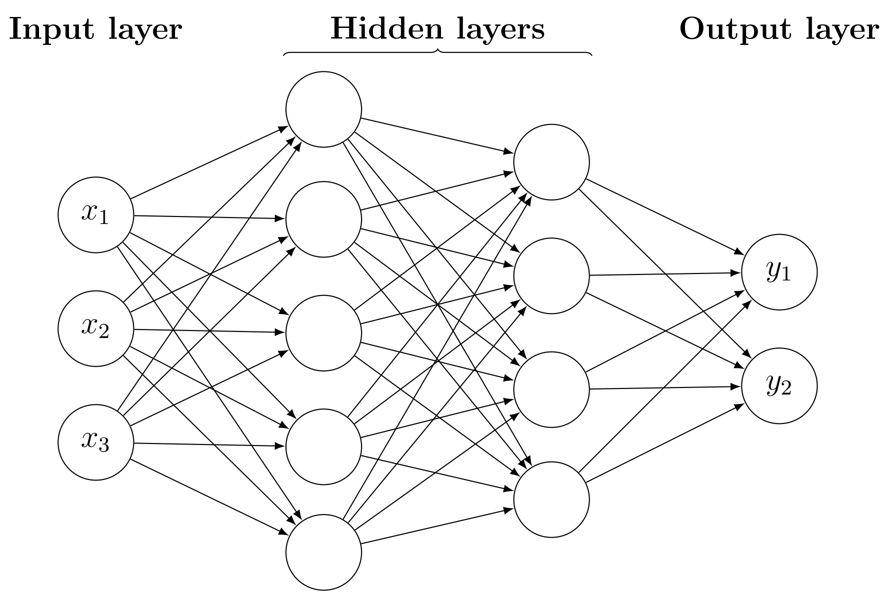

Figure 9.2: Schema of a multi-layer neural network with inputs x_1, x_2, x_3, outputs y_1, y_2 and two hidden layers

arbitrarily small precision by a SLN with finite number of neurons and nonconstant, bounded, and monotonically-increasing continuous activation function.

Consider such an SLN consisting of one hidden layer $l = 1$ with k_l neurons, output layer $o = l+1$ with k_o neurons and weights $\boldsymbol{w}$. Let us partition the weights in $\boldsymbol{w}$ into a collection of weight vectors corresponding to their respective neurons. Mathematically, let $\mathbf{w}_{l,1}, \mathbf{w}_{l,2}, \ldots, \mathbf{w}_{l,k_l}$ be weight vectors for neurons in hidden layer such that $(\boldsymbol{w}_{l,i})_j$ represents the weight from the j-th neuron in the input layer to the i-th neuron in the hidden layer. For the neurons in the output layer let $\mathbf{w}_{o,1}, \mathbf{w}_{o,2}, \ldots, \mathbf{w}_{o,k_o}$ be weight vectors such that $(\boldsymbol{w}_{o,i})_j$ represents the weight from the j-th neuron in the hidden layer to the i-th neuron in the output layer. Let ϕ_l be the activation function of neurons in the hidden layer and ϕ_o the activation function of neurons in the output layer. Furthermore, let us define matrices

$$\boldsymbol{W}^{(l)} = \begin{pmatrix} \boldsymbol{w}'_{l,1} \\ \boldsymbol{w}'_{l,2} \\ \vdots \\ \boldsymbol{w}'_{l,k_l} \end{pmatrix} \qquad \boldsymbol{W}^{(o)} = \begin{pmatrix} \boldsymbol{w}'_{o,1} \\ \boldsymbol{w}'_{o,2} \\ \vdots \\ \boldsymbol{w}'_{o,k_o} \end{pmatrix}. \tag{9.1}$$

In this representation $\boldsymbol{W}^{(l)}_{i,j} = (\boldsymbol{w}_{l,i})_j$ is the weight from the j-th neuron in the input layer to the i-th neuron in the hidden layer and $\boldsymbol{W}^{(o)}_{i,j} = (\boldsymbol{w}_{o,i})_j$ the weight from the j-th neuron in the hidden layer to the i-th neuron in the output layer.

Given an input vector $\boldsymbol{x}$ the output of this network $\boldsymbol{f_w}$ is then defined as

$$\boldsymbol{f_w}(\boldsymbol{x}) = \boldsymbol{\phi}_o[\boldsymbol{W}^{(o)}\boldsymbol{\phi}_l[\boldsymbol{W}^{(l)}\boldsymbol{x}]].$$

9.3 Training

Suppose we have an MLN with weights $\boldsymbol{w} = (w_1, w_2, \ldots, w_n)$. As before we divide the training process into three phases: initialisation, estimation, and model selection. The initialisation sets the initial value of $\boldsymbol{w}$ and in case on MLN coincides with model selection. The estimation then adjusts the values of weights $\boldsymbol{w}$ in order to approximate correct outputs.

There are two main approaches to estimating the weights of neural networks: supervised and unsupervised learning. The correct approach depends mainly on the type of network, which is influenced by the problem description.

All neural networks in this thesis use supervised learning. Supervised learning uses data separated into pairs $(\boldsymbol{x}, \boldsymbol{d})$ of input $\boldsymbol{x}$ and output vectors $\boldsymbol{d}$. We use the term *samples* to refer to the pairs $(\boldsymbol{x}, \boldsymbol{d})$. The goal of training is to infer the values of weights $\boldsymbol{w}$ so that given any input $\boldsymbol{x}$, the corresponding output of the network $\boldsymbol{f_w}(\boldsymbol{x})$ is equal to $\boldsymbol{d}$.

Since estimating the weights of a neural network is an NP-complete problem [70], it is more practical to use various heuristics to approximate the solution. One of the most popular heuristics used in practice and also the one used for networks in this thesis is the *gradient descent* [71], which we describe in Section 9.3.1. Gradient descent is an optimisation heuristic that minimises a loss function using the gradient of said loss function to tune the weights $\boldsymbol{w}$. Computing the gradient analytically can be very time consuming. However, this process can be significantly sped up using back-propagation algorithm described in Section 9.3.2. Moreover, gradient descent can be further optimised by introducing adaptive learning rates for each weight. Section 9.3.3 describes such a method. Section 9.3.5 is devoted to the initialisation of weights $\boldsymbol{w}$ in order to guide gradient descent to reach better optima and to reach them faster. It also serves as model selection procedure. Lastly, in Section 9.3.4 we explain the common problems of training neural networks and how to deal with them.

Throughout this section we use the term *batch* to refer to a finite set consisting of a number of samples, i.e. $B = \{(\boldsymbol{x}_1, \boldsymbol{d}_1), (\boldsymbol{x}_2, \boldsymbol{d}_2), \ldots, (\boldsymbol{x}_i, \boldsymbol{d}_i), \ldots\}$. For proofs of various claims made in this Section we refer to the book by Nielsen [67].

9.3.1 Gradient descent

The basis of gradient descent is the minimisation of a loss function $\mathcal{L}(\boldsymbol{w}, B)$ parametrised by the network weights $\boldsymbol{w} = (w_1, w_2, \ldots, w_n)$. In this thesis we use *quadratic loss function*, also called *mean squared error* (MSE). MSE is defined on a batch B as

$$\mathcal{L}(\boldsymbol{w}, B) = \frac{1}{2|B|} \sum_{(\boldsymbol{x},\boldsymbol{d}) \in B} (\boldsymbol{d} - \boldsymbol{f_w}(\boldsymbol{x}))^2. \tag{9.2}$$

The loss function is minimised iteratively over all samples $(\boldsymbol{x}, \boldsymbol{d})$ in the batch B. In each iteration, called *epoch*, the weights $\boldsymbol{w}$ are updated in the opposite direction of the gradient of the loss function. In the following we derive the update rule for $\boldsymbol{w}$ in order to minimise $\mathcal{L}(\boldsymbol{w}, B)$.

Let $\boldsymbol{w}_t$ be the weights after t epochs and let $\boldsymbol{w}_{t+1}$ be the weights after $t+1$ epochs. Let $\triangle\boldsymbol{w} = \boldsymbol{w}_{t+1} - \boldsymbol{w}_t$ be the vector of changes in weights and $\triangle\mathcal{L} = \mathcal{L}(\boldsymbol{w}_{t+1}, B) - \mathcal{L}(\boldsymbol{w}_t, B)$ the change in the loss function. The gradient of the loss function with respect to weights $\boldsymbol{w}_t$ is defined as

$$\nabla_{\boldsymbol{w}}\mathcal{L}(\boldsymbol{w}_t, B) = \left(\frac{\partial\mathcal{L}(\boldsymbol{w}_t, B)}{\partial w_1}, \frac{\partial\mathcal{L}(\boldsymbol{w}_t, B)}{\partial w_2}, \ldots, \frac{\partial\mathcal{L}(\boldsymbol{w}_t, B)}{\partial w_n} \right).$$

The change in loss function can be then expressed as a dot product of vector $\triangle \boldsymbol{w}$ and gradient $\nabla_{\boldsymbol{w}}\mathcal{L}(\boldsymbol{w}, B)$:

$$\triangle \mathcal{L} \approx \nabla_{\boldsymbol{w}}\mathcal{L}(\boldsymbol{w}_t, B)' \cdot \triangle \boldsymbol{w}$$

In order to minimise the loss function we want $\triangle \mathcal{L} < 0$. This is satisfied after we let $\triangle \boldsymbol{w} = -\lambda \nabla_{\boldsymbol{w}}\mathcal{L}(\boldsymbol{w}_t, B)$, where $0 < \lambda$ is a parameter called *learning rate*, because $\triangle \mathcal{L} \approx \nabla_{\boldsymbol{w}}\mathcal{L}(\boldsymbol{w}_t, B)'(-\lambda \nabla_{\boldsymbol{w}}\mathcal{L}(\boldsymbol{w}_t, B)) = -\lambda \|\nabla_{\boldsymbol{w}}\mathcal{L}(\boldsymbol{w}_t, B)\|^2 < 0$. The update rule for gradient descent then becomes

$$\boldsymbol{w}_{t+1} \leftarrow \boldsymbol{w}_t - \lambda \nabla_{\boldsymbol{w}}\mathcal{L}(\boldsymbol{w}_t, B). \tag{9.3}$$

Depending on the batch size $|B|$ in each epoch, gradient descent can be divided into three main types:

- *Batch gradient descent*: batch $|B|$ contains all available samples. It is guaranteed to converge to global optimum for convex loss surfaces and local optimum on nonconvex surfaces. It also perform redundant computations on large datasets that contain many similar samples and can get stuck in local optima.
- *Stochastic gradient descent*: batch contains only one randomly chosen sample, i.e. $|B| = 1$. It is usually much faster that batch gradient descent and can escape local optimum and find a better solution. On the other hand, stochastic descent is not guaranteed to reach global optimum.
- *Minibatch gradient descent*: batch contains a number (e.g. 50) of randomly chosen samples. The exact batch size is a hyperparameter than can be optimised. It is faster that batch descent with lesser risk of getting stuck in local optima than stochastic descent. On the other hand, it is not guaranteed to converge to global optimum.

For the neural networks used in this thesis we use minibatch gradient descent with batch size optimised using grid search. The pseudo-code can be seen in Algorithm 2.

Algorithm 2 Minibatch gradient descent optimisation

```
1: initialise network weights $\boldsymbol{w}_0$ with small random values, e.g. $\boldsymbol{w}_0 \sim \mathcal{N}(0, \epsilon)$
2: for epoch $t = 0, 1, \ldots$ do
3:     for each batch $B$ of samples do
4:         for each $(\boldsymbol{x}, \boldsymbol{d}) \in B$ do
5:             compute network output $\boldsymbol{f}_{\boldsymbol{w}_t}(\boldsymbol{x})$ for input $\boldsymbol{x}$
6:         end for
7:         compute loss function $\mathcal{L}(\boldsymbol{w}_t, B)$
8:         update weights $\boldsymbol{w}_{t+1} \leftarrow \boldsymbol{w}_t - \lambda \nabla_{\boldsymbol{w}}\mathcal{L}(\boldsymbol{w}_t, B)$
9:     end for
10: end for
```

So far we ignored the most important part of gradient descent, which is the computation of the very gradient $\nabla_{\boldsymbol{w}}\mathcal{L}(\boldsymbol{w}_t, B)$. Although this can be done analytically, the sheer number of weights in $\boldsymbol{w}$ can render this approach all but impossible. In the Section 9.3.2 we present a fast algorithm, called *back-propagation*, commonly used for gradient computations.

9.3.2 Back-propagation

Back-propagation is an efficient algorithm for computing gradients, popularised by Rumelhart et al. in 1986 [72]. It is based on the *chain rule*, a mathematical formula used for computing the derivative of the composition of two or more functions [73]. Using the Leibniz's notation the chain rule is in the form of

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}$$

where $z = f(y)$ and $y = g(x)$.

Consider a network with input layer, hidden layers $1, 2, \dots, k$ and output layer $o = k + 1$. For the purposes of back-propagation let us separate the network parameters $\boldsymbol{w}$ into weight matrices for hidden layers $\boldsymbol{W}^{(1)}, \boldsymbol{W}^{(2)}, \dots, \boldsymbol{W}^{(k)}$ and weight matrix for output layer $\boldsymbol{W}^{(o)}$ (see Section (9.1)). Each matrix $\boldsymbol{W}^{(l)}$ is a matrix of weights from layer $l - 1$ to layer l where $\boldsymbol{W}^{(l)}_{i,j}$ is the weight from j-th neuron in the layer $l - 1$ to i-th neuron in the layer l. Furthermore, let $\xi_{l,i}$ be the activation of neuron i in layer l and let $\boldsymbol{\xi}_l = (\xi_{l,1}, \xi_{l,2}, \dots, \xi_{l,k_l})$ be the *activation vector* of layer l with k_l neurons. Finally, let ϕ_l be the activation function of layer l and let $\boldsymbol{\phi}_l[\boldsymbol{\xi}_l]$ be the *output vector* of layer l. The update rule from gradient descent (9.3) can be then rewritten into

$$\textbf{for each } \text{layer } l : \ \boldsymbol{W}^{(l)}_{t+1} \leftarrow \boldsymbol{W}^{(l)}_t - \lambda \nabla_{\boldsymbol{W}^{(l)}} \mathcal{L}(\boldsymbol{w}_t, B) \tag{9.4}$$

where $\nabla_{\boldsymbol{W}^{(l)}} \mathcal{L}(\boldsymbol{w}_t, B)$ is a matrix of partial derivatives defined as

$$\nabla_{\boldsymbol{W}^{(l)}} \mathcal{L}(\boldsymbol{w}_t, B)_{i,j} = \left(\frac{\partial \mathcal{L}(\boldsymbol{w}_t, B)}{\partial \boldsymbol{W}^{(l)}_{i,j}} \right)$$

Back-propagation uses chain rule to compute $\nabla_{\boldsymbol{W}^{(l)}} \mathcal{L}(\boldsymbol{w}_t, B)$ for each weight matrix $\boldsymbol{W}^{(l)}$ separately, starting from the output layer and back-propagating through hidden layers until it reaches the input layer.

Consider any sample from batch $(\boldsymbol{x}, \boldsymbol{d}) \in B$. If we define the loss function for this sample as

$$\mathcal{L}(\boldsymbol{w}_t, \boldsymbol{x}, \boldsymbol{d}) = \frac{1}{2}(\boldsymbol{d} - \boldsymbol{f}_{\boldsymbol{w}_t}(\boldsymbol{x}))^2$$

the overall loss from (9.2) becomes

$$\mathcal{L}(\boldsymbol{w}_t, B) = \frac{1}{|B|} \sum_{(\boldsymbol{x}, \boldsymbol{d}) \in B} \mathcal{L}(\boldsymbol{w}_t, \boldsymbol{x}, \boldsymbol{d}).$$

Subsequently, the partial derivative of $\mathcal{L}(\boldsymbol{w}_t, B)$ with respect to weight $\boldsymbol{W}^{(l)}_{i,j}$ is

$$\frac{\partial \mathcal{L}(\boldsymbol{w}_t, B)}{\partial \boldsymbol{W}^{(l)}_{i,j}} = \frac{1}{|B|} \sum_{(\boldsymbol{x}, \boldsymbol{d}) \in B} \frac{\partial \mathcal{L}(\boldsymbol{w}_t, \boldsymbol{x}, \boldsymbol{d})}{\partial \boldsymbol{W}^{(l)}_{i,j}}$$

and as a consequence the gradient $\nabla_{\boldsymbol{W}^{(l)}} \mathcal{L}(\boldsymbol{w}_t, B)$ becomes

$$\nabla_{\boldsymbol{W}^{(l)}} \mathcal{L}(\boldsymbol{w}_t, B) = \frac{1}{|B|} \sum_{(\boldsymbol{x}, \boldsymbol{d}) \in B} \nabla_{\boldsymbol{W}^{(l)}} \mathcal{L}(\boldsymbol{w}_t, \boldsymbol{x}, \boldsymbol{d}). \tag{9.5}$$

This enables us to accumulate the gradient while iterating over a batch of samples B and update the weights only after the iteration has ended.

To compute $\nabla_{\boldsymbol{W}^{(l)}}\mathcal{L}(\boldsymbol{w}_t, \boldsymbol{x}, \boldsymbol{d})$ we utilise the chain rule. One element of matrix $\nabla_{\boldsymbol{W}^{(l)}}\mathcal{L}(\boldsymbol{w}_t, \boldsymbol{x}, \boldsymbol{d})$ then becomes

$$\nabla_{\boldsymbol{W}^{(l)}}\mathcal{L}(\boldsymbol{w}_t, \boldsymbol{x}, \boldsymbol{d})_{i,j} = \frac{\partial \mathcal{L}(\boldsymbol{w}_t, \boldsymbol{x}, \boldsymbol{d})}{\partial \boldsymbol{W}^{(l)}_{i,j}} = \frac{\partial \mathcal{L}(\boldsymbol{w}_t, \boldsymbol{x}, \boldsymbol{d})}{\partial \xi_{l,i}} \frac{\partial \xi_{l,i}}{\partial \boldsymbol{W}^{(l)}_{i,j}}. \tag{9.6}$$

Furthermore, if we define the *error* $\delta^{(l)}_i$ of neuron i in layer l as

$$\delta_{l,i} = \frac{\partial \mathcal{L}(\boldsymbol{w}_t, \boldsymbol{x}, \boldsymbol{d})}{\partial \xi_{l,i}}$$

and use the fact that

$$\frac{\partial \xi_{l,i}}{\partial \boldsymbol{W}^{(l)}_{i,j}} = \phi_{l-1}(\xi_{l-1,j}),$$

instead of (9.6) we can write

$$\nabla_{\boldsymbol{W}^{(l)}}\mathcal{L}(\boldsymbol{w}_t, \boldsymbol{x}, \boldsymbol{d})_{i,j} = \delta_{l,i}\phi_{l-1}(\xi_{l-1,j}). \tag{9.7}$$

Consequently, is we define the *error vector* $\boldsymbol{\delta}_l$ for layer l as

$$\boldsymbol{\delta}_l = (\delta_{l,1}, \delta_{l,2}, \dots, \delta_{l,k}) = \left(\frac{\partial \mathcal{L}(\boldsymbol{w}_t, \boldsymbol{x}, \boldsymbol{d})}{\partial \xi_{l,1}}, \frac{\partial \mathcal{L}(\boldsymbol{w}_t, \boldsymbol{x}, \boldsymbol{d})}{\partial \xi_{l,2}}, \dots, \frac{\partial \mathcal{L}(\boldsymbol{w}_t, \boldsymbol{x}, \boldsymbol{d})}{\partial \xi_{l,k}}\right)$$

we can vectorise Equation (9.7) and express the gradient $\nabla_{\boldsymbol{W}^{(l)}}\mathcal{L}(\boldsymbol{w}_t, \boldsymbol{x}, \boldsymbol{d})$ as

$$\nabla_{\boldsymbol{W}^{(l)}}\mathcal{L}(\boldsymbol{w}_t, \boldsymbol{x}, \boldsymbol{d}) = \boldsymbol{\delta}_l \boldsymbol{\phi}_{l-1}[\boldsymbol{\xi}_{l-1}]'. \tag{9.8}$$

Using the chain rule one can prove that for the output layer o the error vector $\boldsymbol{\delta}_o$ can be computed directly from the network output $\boldsymbol{f}_{\boldsymbol{w}_t}(\boldsymbol{x})$ and for neurons in the hidden layers $l = 1, 2, \dots$ the error vector $\boldsymbol{\delta}_l$ can be obtained from the error vector for the subsequent layer $\boldsymbol{\delta}_{l+1}$. Specifically, one can prove [67] that for the output layer the error vector can be obtained as

$$\boldsymbol{\delta}_o = (\boldsymbol{f}_{\boldsymbol{w}_t}(\boldsymbol{x}) - \boldsymbol{d}) \odot \boldsymbol{\phi}'_o[\boldsymbol{\xi}_o]$$

and the error vector for hidden layer $l = o-1, o-2, \dots, 2, 1$ can be then computed as

$$\boldsymbol{\delta}_l = (\boldsymbol{W}^{(l+1)})'\boldsymbol{\delta}_{l+1} \odot \boldsymbol{\phi}'_l[\boldsymbol{\xi}_l]$$

where ϕ'_o and ϕ'_l are the derivations of activation functions for the output layer o and the hidden layer l, respectively. In the case of sigmoid activation function the derivation can be expressed as $\phi'(\xi) = \phi(\xi)(1 - \phi(\xi))$, in case of tanh activation function it is $\phi'(\xi) = 1 - \phi(\xi)^2$.

The idea behind back-propagation algorithm is that for each sample $(\boldsymbol{x}, \boldsymbol{d})$ in batch B we first compute the vector of activations $\boldsymbol{\xi}^{(l)}$ for all layers l and then compute the network output $\boldsymbol{f}_{\boldsymbol{w}_t}(\boldsymbol{x})$ together with the loss $\mathcal{L}(\boldsymbol{w}_t, \boldsymbol{x}, \boldsymbol{d})$ (forward pass). Then, iterating over the layers backwards from output to input (backward pass), for each layer l we compute the *error vector* $\boldsymbol{\delta}_l$. After backward pass has finished and the error term δ is computed for every neuron in the network, we compute $\nabla_{\boldsymbol{W}^{(l)}}\mathcal{L}(\boldsymbol{w}_t, \boldsymbol{x}, \boldsymbol{d})$ from (9.8), then use it in (9.5) to compute $\nabla_{\boldsymbol{W}^{(l)}}\mathcal{L}(\boldsymbol{w}_t, B)$ and finally update the weights layer by layer using (9.4).

The pseudo-code for minibatch gradient descent with back-propagation for gradient computation is summarised in Algorithm 3

Algorithm 3 Minibatch gradient descent with back-propagation for gradient computation

```
 1: initialise network weights $\boldsymbol{w}_0$ with small random values, i.e. $w_i \sim \mathcal{N}(0, \epsilon)$
 2: for epoch $t = 0, 1, \dots$ do
 3:     for each batch $B$ of samples do
 4:         for each $(\boldsymbol{x}, \boldsymbol{d}) \in B$ do
 5:             perform forward pass to obtain $\boldsymbol{f}_{\boldsymbol{w}_t}(\boldsymbol{x})$ and $\mathcal{L}(\boldsymbol{w}_t, \boldsymbol{x}, \boldsymbol{d})$
 6:             for the output layer $o$ set $\boldsymbol{\delta}_o \leftarrow (\boldsymbol{f}_{\boldsymbol{w}_t}(\boldsymbol{x}) - \boldsymbol{d}) \odot \boldsymbol{\phi}'_o[\boldsymbol{\xi}_o]$
 7:             for $l = o-1, o-2, \dots, 1$ do
 8:                 $\boldsymbol{\delta}_l \leftarrow (\boldsymbol{W}_t^{(l+1)})'\boldsymbol{\delta}_{l+1} \odot \boldsymbol{\phi}'_l[\boldsymbol{\xi}_l]$
 9:                 $\nabla_{\boldsymbol{W}^{(l)}}\mathcal{L}(\boldsymbol{w}_t, \boldsymbol{x}, \boldsymbol{d}) \leftarrow \boldsymbol{\delta}_l \boldsymbol{\phi}_{l-1}[\boldsymbol{\xi}_{l-1}]'$
10:             end for
11:         end for
12:         for each layer $l$ do
13:             $\nabla_{\boldsymbol{W}^{(l)}}\mathcal{L}(\boldsymbol{w}_t, B) \leftarrow \frac{1}{|B|}\sum_{(\boldsymbol{x},\boldsymbol{d})\in B} \nabla_{\boldsymbol{W}^{(l)}}\mathcal{L}(\boldsymbol{w}_t, \boldsymbol{x}, \boldsymbol{d})$
14:             $\boldsymbol{W}_{t+1}^{(l)} \leftarrow \boldsymbol{W}_t^{(l)} - \lambda \nabla_{\boldsymbol{W}^{(l)}}\mathcal{L}(\boldsymbol{w}_t, B)$
15:         end for
16:     end for
17: end for
```

9.3.3 Adaptive moment estimation

While using minibatch gradient descent in practice, one may encounter a number of challenges. Firstly, it may be difficult to chose a suitable learning rate λ, as too small a λ leads to very slow convergence to optimum while λ set too high can result in oscillations around the optimum or even divergence from it. Additionally, the same learning rate λ applies to all parameters in $\boldsymbol{w}$ to the same extent. However, the nature of inputs $x_1, x_2, \dots, x_n$ of the input vector $\boldsymbol{x}$ and their influence on the output $\boldsymbol{f}_{\boldsymbol{w}}(\boldsymbol{x})$ may differ significantly.

For these reason we use adaptive moment estimation (ADAM) [74] for neural networks in this thesis. Apart from describing the method, in the paper the authors of ADAM compare it to other common gradient descent modifications and ADAM comes on top.

ADAM tackles the problem of minibatch gradient descent by including separate learning rates for each weight in $\boldsymbol{w}$. Furthermore, these learning rates are not static but adaptive, meaning they change with the gradient.

This is done by keeping track of an exponentially decaying moving average of past gradients

$$\boldsymbol{g}_t = \beta_1 \boldsymbol{g}_{t-1} + (1-\beta_1)\nabla_{\boldsymbol{w}}\mathcal{L}(\boldsymbol{w}_t, B)$$

and also exponentially decaying moving average of past squared gradients

$$\boldsymbol{v}_t = \beta_2 \boldsymbol{v}_{t-1} + (1-\beta_2)\nabla_{\boldsymbol{w}}(\mathcal{L}(\boldsymbol{w}_t, B))^2$$

where $0 < \beta_1, \beta_2 < 1$ are parameters and $(\mathcal{L}(\boldsymbol{w}_t, B))^2 = \mathcal{L}(\boldsymbol{w}_t, B) \odot \mathcal{L}(\boldsymbol{w}_t, B)$. Vector $\boldsymbol{g}_t$ is the estimate of the mean (first moment) and vector $\boldsymbol{v}_t$ is the estimate of the variance (second moment) of the gradients. Authors suggest that the best values for parameters are $\beta_1 = 0.9$ and $\beta_2 = 0.999$.

The initial values are $\boldsymbol{g}_0 = \mathbf{0}_k$ and $\boldsymbol{v}_0 = \mathbf{0}_k$. This means that during the first couple of epochs the estimates $\boldsymbol{g}_t$ and $\boldsymbol{v}_t$ are biased towards zero. Authors of ADAM mitigate this issue by computing bias-corrected first and second moments estimates

$$\begin{aligned} \boldsymbol{g}_t^* &= \frac{\boldsymbol{g}_t}{1-\beta_1^t} \\ \boldsymbol{v}_t^* &= \frac{\boldsymbol{v}_t}{1-\beta_2^t} \end{aligned}$$

where β_1^t, β_2^t denote β_1, β_2 raised to the power of t.

Bias-corrected first and second moments estimates are then used to update the weights using the following rule:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \lambda \frac{\boldsymbol{g}_t^*}{\sqrt{\boldsymbol{v}_t^*} + \epsilon}$$

where λ and ϵ are parameters for which authors recommend values $\lambda = 0.001$ and $\epsilon = 10^{-8}$.

9.3.4 Generalisation and overfitting

One of the most important aspects of the quality of machine learning models, including neural networks, is their ability to *generalise* [13]. Generalisation is the ability of a model to perform well on insofar unseen instances beyond the data it was trained on.

The generalisation abilities of a machine learning model are usually measured by splitting all the available data into disjunct *training set* T and *validation set* V. In the case of neural networks both T and V are sets consisting of samples $(\boldsymbol{x}, \boldsymbol{d})$. Model is then trained using only data from the training set. The generalisation abilities of the model are evaluated by comparing the performance on the training set with the performance on the validation set.

Ideally, the model is able to perform as well on the validation set as on the training set. However, in some cases it learns not only the general patterns but also the random noise present in the training set. In machine learning this phenomenon is refereed to as *overfitting*. Overfitting with respect to neural networks can be defined as the situation in which for network with weights $\boldsymbol{w}$ there exist weights $\boldsymbol{w}^*$ such that $\mathcal{L}(\boldsymbol{w}, T) \leq \mathcal{L}(\boldsymbol{w}^*, T)$ but $\mathcal{L}(\boldsymbol{w}, V) > \mathcal{L}(\boldsymbol{w}^*, V)$ [13].

A common overfitting prevention technique is *early stopping* [75]. While the network is trained on the training set T, after each epoch t both $\mathcal{L}(\boldsymbol{w}_t, T)$ and $\mathcal{L}(\boldsymbol{w}_t, V)$ are evaluated. The training is stopped when $\mathcal{L}(\boldsymbol{w}_t, V)$ increases during each of k consecutive epochs, where k is a hyperparameter subject to optimisation.

9.3.5 Initialisation and model selection

Random weight initialisation performed at the beginning of gradient descend algorithms can have significant impact on both the speed of convergence and the quality of found local optima as demonstrated by Glorot and Bengio [76].

For this reason we consider restarting neural network with the same topology and hyperparameters multiple times. After every network is trained on training set T, their loss function $\mathcal{L}$ is evaluated on the validation set V and the best network (lowest $\mathcal{L}$) is chosen to make forecast (see Section 10.2). For this reason the initialisation of MLN in fact coincides with model selection.

Glorot and Bengio in [76] also introduce a new random initialisation scheme and compare it to others. Their scheme performs the best and for this reason we chose to also use it. Weights in layer l are drawn from uniform distribution $\mathcal{U}$ with the interval scaled with respect to the number of neurons in the preceding layer $l-1$ and layer l. Mathematically, for initial weight matrix $\boldsymbol{W}_0^{(l)}$ the random initialisation scheme can be described as

$$\boldsymbol{W}_0^{(l)} \sim \mathcal{U}\left(-\sqrt{\frac{6}{i+j}}, \sqrt{\frac{6}{i+j}}\right)$$

where i is the number of neurons in layer l and j the number of neurons in layer $l-1$.

9.4 Time series data

The goal of an MLN is, after being trained on a set of samples $(\boldsymbol{x}, \boldsymbol{d})$, to produce correct $\boldsymbol{f_w}(\boldsymbol{x}^*)$ given an insofar unseen input $\boldsymbol{x}^*$. Traditionally, each feature x_i of the inputs is considered to be drawn independently from some probability distribution. However, in the case of time series data the independence property is violated as there is a significant correlation between consecutive observations.

For this reason MLNs use the *rolling window* method [77] when dealing with time series data $\boldsymbol{y}_t$. The method can be described in the following steps:

(1): The values of *window size* w, *window shift* s and forecast horizon h are chosen. Window size w is a hyperparameter that needs to be estimated prior to training, but window shift s and forecast horizon h are usually problem specific.

(2): $\boldsymbol{y}$ is transformed into a set of samples $(\boldsymbol{y}_{si+1:si+w}, \boldsymbol{y}_{si+w+1:si+w+h})$, where $i = 0, 1, \ldots$.

(3): The samples are separated into training and validation sets.

(4): The network is trained on the training set and its loss evaluated on the validation set until at least one of the stopping criteria is met.

(5): Forecasts are produced as $\hat{\boldsymbol{y}}_{t+h|t} = \boldsymbol{f}(\boldsymbol{y}_{t-w+1:t})$.

It may also be beneficial to encode the date as part of the inputs. This is done using *dummy variables*. A dummy variable takes values 0 or 1 to indicate the presence or absence of some categorical effect. In our case the categories include day of the week (Sunday, Monday, ...) and month (January, February, ...). This means 19 additional inputs.

In some cases it may be beneficial to include other time series (weather data) as part of the input, akin to predictors or explanatory variables in regression. In our case, since historical forecast were unavailable to us, we used 48 real weather observations corresponding to each half-hour of the day for which the network is supposed to forecast energy consumption. Mathematically, $\boldsymbol{x}_T$ with $T > t$ is time series for a weather attribute (e.g. temperature, humidity or wind speed) and we want to forecast load profile $\hat{\boldsymbol{y}}_{t+h|t}$, the network input consists of concatenated vectors $\boldsymbol{x}_{t+1:t+h}$ and $\boldsymbol{y}_{t-w+1:t}$. Note that this undoubtedly introduces some bias to our forecasts.

10. Methodology

The focus of this chapter is the methodology used while conducting experiments and comparing imputation and forecasting models.

This chapter is divided into three sections. In Section 10.1 we list various accuracy measures used to compare both imputation and forecasting models. This is followed by Section 10.2 where we describe how the data is split into training and testing sets and how the models are evaluated. Lastly, in Section 10.3 we list a number of preprocessing methods designed to extract or separate useful information even before the training process.

10.1 Accuracy measures

Our primary concern is to compare models using the accuracy of their forecasts. It is difficult to define accuracy as the specific purpose of forecasting may dictate what aspects of forecasts are more important than others. Therefore, in this section we present four accuracy measures that we used jointly to compare the models.

In all of the accuracy measures we use $\hat{\boldsymbol{y}}$ to refer to forecasts produced by the model under consideration and $\boldsymbol{y}$ to denote the true observations. It is also important to note that since energy consumption of a household cannot be negative, we replaced all negative values in $\hat{\boldsymbol{y}}$ by zeros.

10.1.1 SRMSE

SRMSE (Scaled RMSE) is derived from RMSE (root mean squared error), which measures accuracy as a standard deviation of forecasts when compared to the actual values:

$$\mathrm{RMSE}(model) = \sqrt{E((\boldsymbol{y} - \hat{\boldsymbol{y}})^2)}$$

RMSE is a very popular measure, largely due to its relevance in the theory of statistical modelling [78].

One disadvantage of RMSE is that it is scale-dependant and cannot be used to compare models used on different datasets or differently preprocessed datasets. For this reason we adopt a scaled variant of RMSE called SRMSE:

$$\mathrm{SRMSE}(model) = \frac{\mathrm{RMSE}(model)}{E(\boldsymbol{y})}$$

which has the range of $(0, \infty)$ and does not have this shortcoming.

10.1.2 SMAPE

SMAPE (symmetric MAPE) is a modification of MAPE (mean absolute percentage error), which measures accuracy as a ratio of errors to the actual value:

$$\mathrm{MAPE}(model) = E\left(\frac{|\boldsymbol{y} - \hat{\boldsymbol{y}}|}{|\boldsymbol{y}|}\right)$$

with the range of $(0, \infty)$.

MAPE has the advantage of being scale-independent, which means it can be used to compare models across various datasets or preprocessing methods. On the other hand, MAPE has several drawbacks. The first one is that it cannot handle zero values in the data. Secondly, for any small values close to zero MAPE tends to explode. Furhermore, Armstrong [79] brought attention the intrinsic bias of MAPE. It turns out that MAPE favours forecasts that are below the actual value $\hat{y}_t < y_t$ compared to the forecasts above it $\hat{y}_t > y_t$.

SMAPE is a common modification of MAPE designed to remove this bias. It can be expressed as

$$\text{SMAPE}(\mathit{model}) = E\left(\frac{|\boldsymbol{y} - \hat{\boldsymbol{y}}|}{|\boldsymbol{y}| + |\hat{\boldsymbol{y}}|}\right)$$

with the range of $(0, 1)$.

While still being scale-independent, SMAPE also inherits other problems of MAPE. Although less severe, forecasting small values around zero can cause destabilisation of SMAPE value by a division by a number close to zero, because if y_t is close to zero, $\hat{y}_t$ is also likely to be close to zero.

10.1.3 SMAE

SMAE (scaled MAE) is based on MAE (maximum absolute error) that simply records the highest absolute error encountered during forecasting:

$$\text{MAE}(\mathit{model}) = \max(|\boldsymbol{y} - \hat{\boldsymbol{y}}|).$$

To compare models across differently scaled datasets, we modified MAE by scaling it with respect to mean of time series $\boldsymbol{y}$:

$$\text{SMAE}(\mathit{model}) = \frac{\text{MAE}(\mathit{model})}{E(\boldsymbol{y})}$$

with the range of $(0, \infty)$.

10.1.4 MASE

MASE (mean absolute scaled error) was proposed by Hyndman and Koehler [78] to solve common problems the other accuracy measures face. MASE scales the forecast errors based on mean absolute error of a *seasonal naive forecasting method.* Forecasts of seasonal naive forecasting method, denoted as $\hat{\boldsymbol{y}}^*$, are just observations made exactly one season ago, i.e. $\hat{y}^*_t = y_{t-m}$ where m is the frequency of $\boldsymbol{y}$. MASE can then be expressed as

$$\text{MASE}(\mathit{model}) = \frac{E(|\boldsymbol{y} - \hat{\boldsymbol{y}}|)}{E(|\boldsymbol{y} - \boldsymbol{y}^*|)}$$

with the range of $(0, \infty)$. $MASE < 1$ means that the method under consideration is better than the seasonal naive method while $MASE > 1$ indicates the opposite.

The advantages of MASE is that it is scaled, unbiased and it has no problems with values near zero.

When there are multiple seasonalities in the data as is the case with energy consumption, there are also multiple seasonal naive forecasting methods to consider. In order to select the best one, we compare MASE for methods with daily $m = 48$ and weekly $m = 7 \cdot 48$ seasonalities against each other and list the MASE results in Table 10.1. SRMSE, SMAPE and MAE are summarised in Table 10.2.

Table 10.1: MASE of naive methods

true / forecast	daily	weekly
daily	1	1.0439
weekly	0.9776	1

Table 10.2: Experimental results for naive forecasting methods

frequency	SRMSE	SMAPE	SMAE
daily $m = 48$	0.9567	0.2938	6.9203
weekly $w = 7 \cdot 48$	0.9356	0.2945	7.0187

"wa", "ha", "bc" and "des" indicate the type of preprocessing used (week adjustement, hour adjustement, Box-Cox transformation, deseasonalisation).
Results are sorted in ascending order first by SRMSE.

It is clear that forecasts using weekly seasonality are better according to SRMSE and MASE and worse according to SMAPE and SMAE. From this point forward we chose weekly seasonality for evaluating other models.

10.2 Time series cross-validation

Forecasting energy consumption of a real house requires a model to be able to generalise beyond the data used to train it and effectively adapt to unforeseen conditions. If a model performed well on training data but fails to archive comparable performance while incorporated into a real household, then it is overfitted compared to a different model that does worse on training data but better in real situations [13].

One of the most intuitive approaches to combat the overfitting problem is called *cross-validation* [80]. Cross-validation splits the data randomly into two parts: training set and validation set. Training set is used to train various models, the performance of which it subsequently evaluated on the validation set. The drawback of this approach is that because a big portion of data is dedicated to validation, the overall performance of model may be negatively impacted as there are not enough training samples. On the other hand, if the validation set is too small, it may fail to fairly represent the statistical properties of the whole dataset and result in poor estimates of performance.

A popular technique to better utilise available data is a *k-fold cross-validation* [61]. It works by splitting data randomly into k equal-sized partitions and then performing k rounds of training. In each round one partition is chosen to act as the validation set and the other $k-1$ partitions are used for training. However, the random nature of splitting in k-fold cross-validation is not suitable for time series data, as it is not being drawn independently from the same probability distribution but rather exhibits a significant sequential correlations between consecutive observations.

Correlation of consecutive observation is taken into account by the *rolling window* method [77]. First, the size of a window w and forecast horizon h is chosen. Then the models are trained on the first w samples $\boldsymbol{y}_w$ and forecast $\hat{\boldsymbol{y}}_{w+h|w}$. The process continues by shifting the window by s observations, training on $\boldsymbol{y}_{s+1:s+w}$ and forecasting $\hat{\boldsymbol{y}}_{s+w+h|s+w}$. The process is repeated until there is nothing left to forecast. The performance of models is evaluated by comparing the forecasts accumulated during the process with actual observations.

Time series cross-validation used in this thesis is a combination of k-fold cross-validation and rolling window technique and can be described as follows. The initial split of data consists of training time series $\boldsymbol{y}_t$ and testing time series $\boldsymbol{y}_{t+1:T}$. The training time series contain observation for the first three years (1071 days to be exact, i.e. $t = 1071 \cdot 48$), while the testing time series comprises the last year (precisely 369 days, i.e. $T = (1071 + 369) \cdot 48$). The testing time series is subsequently split into equal-sized (except possibly the last one) partitions $\boldsymbol{y}_{t+ikh+1:t+ikh+kh}$ where $i = 0, 1, \ldots$ is the partition number and k denotes the number of days in a partition. Depending on whether one partition represents one week ($k = 7$) or one month ($k = 28$), we refer to it as *weekly cross-validation* or *monthly cross-validation.* Since we are interested in day-ahead forecasts, let the forecast horizon be equal to the number of observations per day, i.e. $h = 48$. Cross-validation continues by iterating over these partitions, at each step i training the model on $\boldsymbol{y}_{t+ikh}$ and forecasting $\hat{\boldsymbol{y}}_{t+ikh+kh|t+ikh}$ day by day. After forecasts $\hat{\boldsymbol{y}}_{t+idh+h|t+idh}$ for a day $d = 0, 1, \ldots, k-1$ are produced, the model is filtered on the real observation $\boldsymbol{y}_{t+idh+1:t+idh+h}$ (see Section 5.5). For MLN filtering is ignored and may be the subject of future work. The training procedure executed at each step i is described in Section 6.3 for ES models, Section 7.3 for ARMA models, Section 8.4 for ARMAX models and Section 9.3 for MLN models. The entire process of time series cross-validation is summarised in Algorithm 4. The best model in line 10 refers to the model with lowest AICc in case of ES (Section 6.3.3), ARMA (Section 7.3.3) and ARMAX (Section 8.4.3) models or network with the lowest loss (Section 9.3.5).

This procedure accomplishes two things. Not only does it help prevent overfitting, but is also simulates how would models be applied in real households. They would be pretrained in a factory using some template data and then integrated into a household where they would be forced to adapt to real conditions by retraining themselves.

10.3 Preprocessing

Energy consumption data can often contain complex trend and seasonal patterns that cannot be easily captured using mathematical models. Some models may even make some assumptions about the data that are not generally true. Other models may become biased because of different scales of observations' attributes.

Algorithm 4 Time series cross-validation

```
1: $T$ : total number of observations in time series
2: $t$ : number of observations for training set
3: $k$ : number of days (forecast horizons) in one partition
4: $h$ : forecast horizon
5: the data is split into training $\boldsymbol{y}_t$ and testing $\boldsymbol{y}_{t+1:T}$ time series
6: testing time series $\boldsymbol{y}_{t+1:T}$ is split into partitions $i = 0, 1, \ldots$, each partition $i$
   is in the form of $\boldsymbol{y}_{t+ikh+1:t+ikh+kh}$
7: for each partition $i = 0, 1, \ldots$ do
8:     train models on the training set $\boldsymbol{y}_{t+ikh}$ and select the best model
9:     for each day $d = 0, 1, \ldots, k-1$ in partition $\boldsymbol{y}_{t+ikh+1:t+ikh+kh}$ do
10:        use the best model to forecast load profile $\hat{\boldsymbol{y}}_{t+idh+h|t+idh}$
11:        the model is filtered on the real load profile $\boldsymbol{y}_{t+idh+1:t+idh+h}$
12:    end for
13:    add partition $\boldsymbol{y}_{t+ikh+1:t+ikh+kh}$ to the training set
14: end for
```

Preprocessing of the historical data before analysis can mitigate these problems [9]. Generally, the purpose of preprocessing the data is to simplify the patterns contained within the data by removing known sources of variation or by making the patterns more consistent across the whole data set. Simple patters are more easily reproduced in mathematical models which leads to better forecasts.

In this section we describe the preprocessing methods used in this thesis. Their impact on a particular model can be inferred from the respective experimental results.

Generally, preprocessing works by extracting various parameters from the data and using them to transform the data and also revert the transformation. Since the data is usually split into training set and testing set, preprocessing the data before this split happens may introduced unwanted bias to the testing set. For this reason the time series cross-validation algorithm 4 is modified to include preprocessing at every step:

All preprocessing methods listed in this section follow the aforementioned procedure. The only exceptions are aggregation (Section 10.3.1) and time adjustment (Section 10.3.2) for obvious reasons.

10.3.1 Aggregation

Our energy consumption data is of a very fine granularity: one observation per minute. However, optimisation algorithms for load shifting usually operate on longer time itervals, i.e. fifteen minutes, half-hour or hour [81]. Since the available weather data is sampled at a rate of one observation per half-hour, we decided to aggregate the minute intervals into half-hour intervals by averaging the values. This means that each day consists of 48 time intervals that need to be forecast simultaneously from previous days. A sample of the data can be seen in Table 10.3. Each column represents the average energy consumption of the next half-hour and each row corresponds to one day.

Aggregation is performed right after imputation. Before aggregating we also removed observation related to incomplete first and last days.

Algorithm 5 Time series cross-validation with preprocessing

```
1: $T$ : total number of observations in time series
2: $t$ : number of observations for training set
3: $k$ : number of days (forecast horizons) in one partition
4: $h$ : forecast horizon
5: the data is split into training $\boldsymbol{y}_t$ and testing $\boldsymbol{y}_{t+1:T}$ time series
6: testing time series $\boldsymbol{y}_{t+1:T}$ is split into partitions $i = 0, 1, \ldots$, each partition $i$
   is in the form of $\boldsymbol{y}_{t+ikh+1:t+ikh+kh}$
7: for each partition $i = 0, 1, \ldots$ do
8:     preprocessing parameters $\boldsymbol{p}$ are extracted from training set $\boldsymbol{y}_{t+ikh}$
9:     preprocess training set $\boldsymbol{y}_{t+ikh}$ using $\boldsymbol{p}$ to produce $\boldsymbol{y}^*_{t+ikh}$
10:    preprocess partition $\boldsymbol{y}_{t+ikh+1:t+ikh+kh}$ using $\boldsymbol{p}$ to produce $\boldsymbol{y}^*_{t+ikh+1:t+ikh+kh}$
11:    train models on the preprocessed training set $\boldsymbol{y}^*_{t+ikh}$ and select the best
       model
12:    for each day $d = 0, 1, \ldots, k-1$ in partition $\boldsymbol{y}^*_{t+ikh+1:t+ikh+kh}$ do
13:        use the best model to forecast load profile $\hat{\boldsymbol{y}}^*_{t+idh+h|t+idh}$
14:        transform $\hat{\boldsymbol{y}}^*_{t+idh+h|t+idh}$ back into $\hat{\boldsymbol{y}}_{t+idh+h|t+idh}$ using parameters $\boldsymbol{p}$
15:        the model is filtered on the preprocessed load profile $\boldsymbol{y}^*_{t+idh+1:t+idh+h}$
16:    end for
17:    add partition $\boldsymbol{y}_{t+ikh+1:t+ikh+kh}$ to the training set
18: end for
```

Table 10.3: A sample of aggregated data

time date	00:00	00:30	01:00	01:30	02:00	...
2008-10-06	0.3516	0.4052	0.4195	0.3289	0.3545	
2008-10-07	0.3790	0.3412	0.2966	0.3959	0.3116	
2008-10-08	0.3040	0.3474	0.3456	0.3111	0.3519	
2008-10-09	1.1834	1.1665	0.3915	0.2904	0.3227	
2008-10-10	1.4949	0.6381	0.6677	0.6755	0.6375	
2008-10-11	0.4051	0.4123	0.3511	0.3665	0.3759	
2008-10-12	0.4015	0.3429	0.3683	0.7925	0.7423	
2008-10-13	0.3840	0.2885	0.3233	0.3362	0.2981	
2008-10-14	0.4074	0.2830	0.3251	0.2813	0.3259	
2008-10-15	0.3163	0.3678	0.3288	0.3252	0.4018	

Each column represents the average energy consumption for the next half-hour, i.e. average consumption between 00:00-00:30 is recorded in column 00:00.
Each row corresponds to one day.

10.3.2 Time adjustement

Some of the models considered in this thesis, namely ES and ARMA models, are able to account for only one type of seasonality. For this reason we introduce two types of time adjustment preprocessing methods.

The first type, referred to as *hour adjustment*, concerns daily seasonality where the data is split into 48 separate time series corresponding to 48 time intervals in each day. This means that each one of these 48 time series contains only observations for the same time interval across all days, e.g. only observations for time interval 10:30-11:00 etc. A separate model is trained on each one of these 48 time series and forecasts for the next day are made using all of them simultaneously with forecast horizon $h = 1$.

The second type is *week adjustment* and it deals with weekly seasonality. The data is split into seven separate time series according to week days, e.g. one of these splits contains only observations recorded on Mondays etc. A separate model is trained on each one of these splits. Forecasts are then made using the appropriate model with forecast horizon $h = 48$.

We can also consider a combination of these two adjustments. This results in $7 \cdot 48 = 336$ different models with forecast horizon $h = 1$.

When no time adjustment is performed, we refer to the data as *unadjusted*.

10.3.3 Box-Cox transformation

Box-Cox transformation [82] is motivated by the need of linear models for data to be normally distributed [83]. It can also help with nonstationary time series by stabilizing the variance.

Box-Cox transformation is a family of power transformations. Its aim is to ensure the observations are normally distributed. Given time series $\boldsymbol{y}$, the transformation to $\boldsymbol{y}^{(\lambda)}$ can be expressed as

$$y_t^{(\lambda)=} \begin{cases} \frac{y_t^\lambda - 1}{\lambda} & \lambda \neq 0 \\ \ln y_t & \lambda = 0 \end{cases}$$

where $\lambda \in [-5, 5]$ is the only parameter.

In this thesis λ is estimated using maximum likelihood estimation of a linear regression model

$$y_t^{(\lambda)} = \beta_0 + \beta_1 y_t.$$

λ resulting in a model with the highest likelihood is the one chosen for transformation. AICc is not necessary as all of the models considered have the same number of parameters.

After a model for forecasting energy consumption is trained on the transformed data, it is necessary to perform an inverse transformation on its forecast $\hat{y}^*_{t+i|t}$ to obtain interpretable results $\hat{y}_{t+i|t}$:

$$\hat{y}_{t+i|t} = \begin{cases} (\lambda(\hat{y}^{(\lambda)}_{t+i|t})^\lambda + 1)^{1/\lambda} & \lambda \neq 0 \\ \exp((\hat{y}^{(\lambda)}_{t+i|t})^\lambda) & \lambda = 0 \end{cases}$$

Some data cannot be power-transformed to be normal. Despite this, Draper and Cox [84] concluded that Box-Cox transformation can still be used to find a transformation that will approximately normalise the data.

Figure 10.1 contains an illustration what Box-Cox transformed energy consumption looks like. The plots look similar but notice how transformed load are centred around zero. Figure 10.2 displays the histogram of energy consumption data after Box-Cox transformation and a red line representing the bimodal distribution fitted to the histogram.

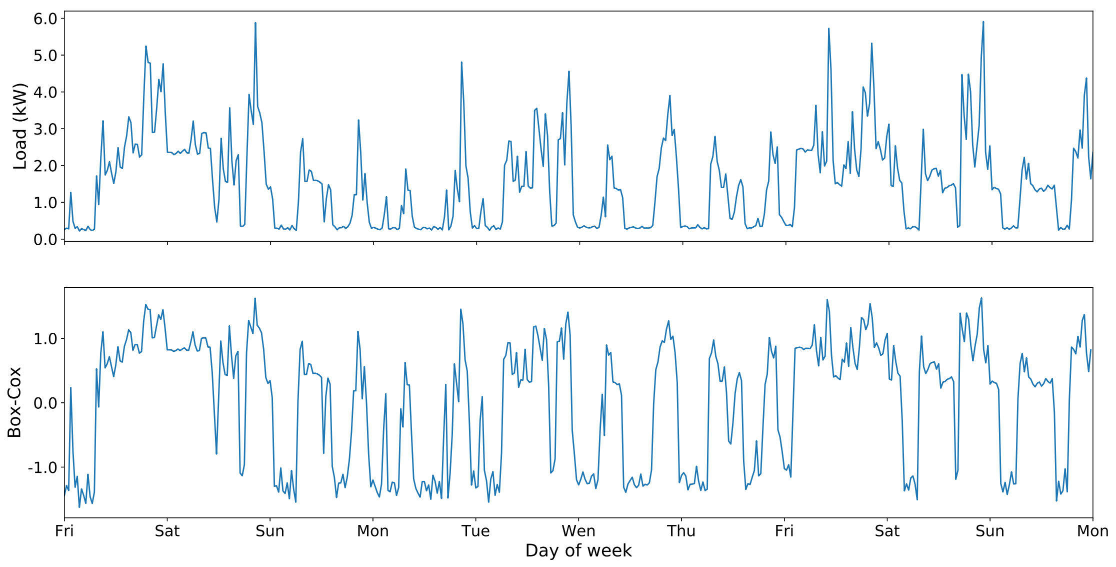

Figure 10.1: Box-Cox transformation of a couple of days

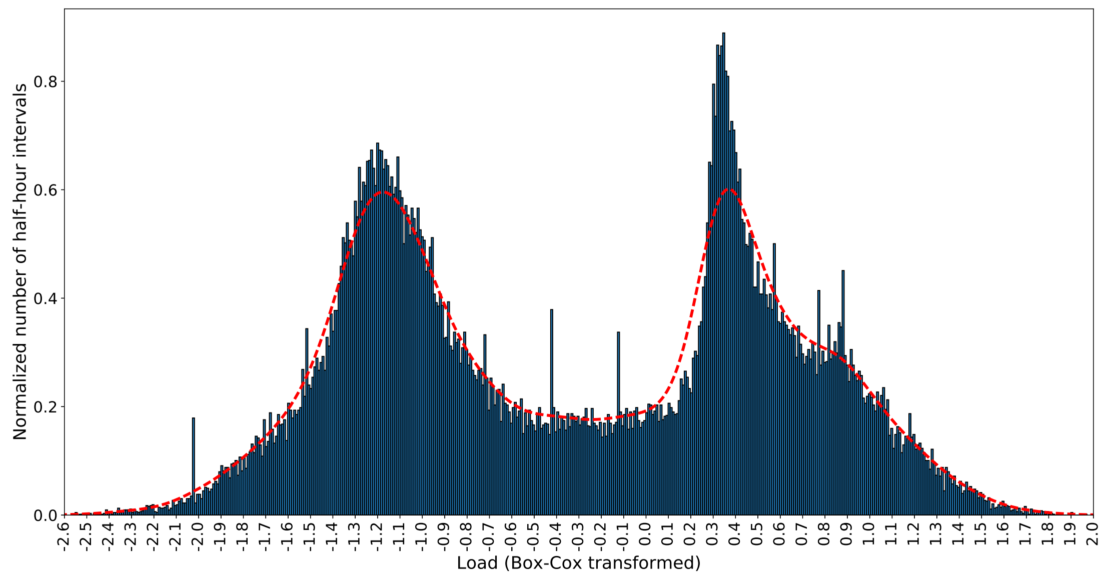

Figure 10.2: Histogram of data after Box-Cox transformation. Red line represents the bimodal distributions that we fit to the data

10.4 Deseasonalisation

ES models and ARMA models can incorporate only one type of seasonality into their forecasts. However, energy consumption data usually contain more than one source of seasonality, namely daily, weekly and annual seasonalities. One can try to use decomposition to remove some of them or at least simplify the data.

In this thesis we use STL [22], a seasonal-trend decomposition procedure based on LOESS (LOcal regrESSion). It decomposes time series $\boldsymbol{y}$ into trend component

$\boldsymbol{b}$, seasonality component $\boldsymbol{s}$ and remainder $\boldsymbol{r}$ so that

$$\mathrm{STL}(\boldsymbol{y}) = \boldsymbol{b} + \boldsymbol{s} + \boldsymbol{r}.$$

LOESS, sometimes called a locally weighted polynomial regression, is a technique that approximates each observation y_t by a regression polynomial $f(t)$ of a small degree, particularly linear polynomials in case of STL decomposition. What distinguishes LOESS from regular regression is fitting $f(t)$ within a window $t - w, t + w$ using weighted least squares, giving more weight to points closer to t and less weight to those far away. We use the term *smoothing window* to refer to the value of w and $\mathbf{LOESS}_w(\boldsymbol{y})$ to denote the result of smoothing time series $\boldsymbol{y}$.

STL works by updating estimates of trend $\boldsymbol{b}^{(i)}$ and seasonal $\boldsymbol{s}^{(i)}$ components iteratively. The detailed steps of STL decomposition procedure are summarised in Algorithm 6.

Algorithm 6 STL decomposition

```
 1: $m \leftarrow$ frequency of $\boldsymbol{y}$
 2: choose smoothing window values $w_l, w_b, w_s$
 3: initialise $\boldsymbol{b}^{(0)} \leftarrow \mathbf{0}$
 4: for each $i = 0, 1, \ldots$ do
 5:     $\boldsymbol{y}^* \leftarrow \boldsymbol{y} - \boldsymbol{b}^{(i)}$                    ▷ compute detrended time series
 6:     $\boldsymbol{s} \leftarrow \mathbf{LOESS}_{w_s}(\boldsymbol{y}^*)$
 7:     $\boldsymbol{l} \leftarrow \mathbf{LOESS}_{w_l}(\mathbf{MAS}_{3\times m\times m}(\boldsymbol{s}))$    ▷ $m\times m\times 3$ moving average smoother
                                                   followed by LOESS regression
 8:     $\boldsymbol{s}^{(i+1)} = \boldsymbol{s} - \boldsymbol{l}$                     ▷ new estimate of seasonal component
 9:     $\boldsymbol{y}^* = \boldsymbol{y} - \boldsymbol{s}^{(i+1)}$                   ▷ compute deseasonalised time series
10:     $\boldsymbol{b}^{(i+1)} \leftarrow \mathbf{LOESS}_{w_b}(\boldsymbol{y}^*)$             ▷ new estimate of trend component
11: end for
```

Authors of STL decomposition [22] found out that the convergence of trend and seasonal components in Algorithm 6 is very fast so two passes of the loop is enough. On authors recommendation we use $w_l = 43$ to prevent competition between the trend and seasonal components for the same source of variance, $w_b = 73$ and $w_s = 7$.

We apply decomposition at each step of cross-validation to deseasonalise the data and train models only on the trend and remainder $\boldsymbol{b}_t + \boldsymbol{r}_t$. Models are then used to forecast load profiles $\hat{\boldsymbol{y}}^*_{t+h|t}$ and these are then combined with the seasonality from previous season $\boldsymbol{s}_{t-m+1:t-m+h}$ to form interpretable forecasts $\hat{\boldsymbol{y}}_{t+h|t} = \hat{\boldsymbol{y}}^*_{t+h|t} + \boldsymbol{s}_{t-m+1:t-m+h}$.

In Figure 10.3 we present an STL decomposition of a couple of days in March 2006. Notice how trend component picks up on larger loads during weekends and seasonality component approximates morning and afternoon load spikes during a business days.

10.4.1 Standardisation

The performance and learning rate of neural networks can be significantly improved by standardisation [85]. Standardisation transforms data into a common

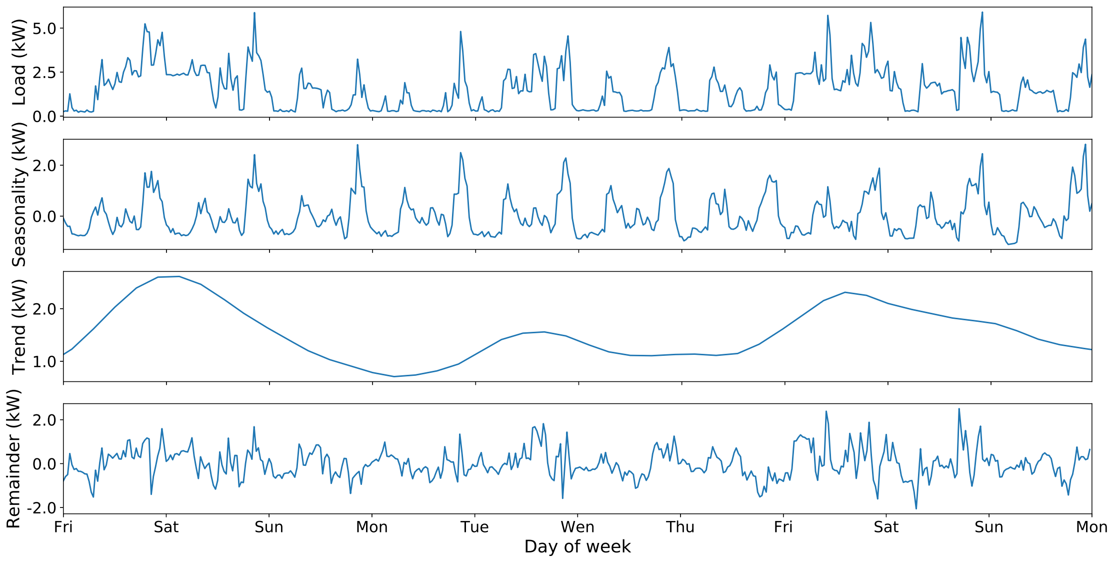

Figure 10.3: STL decomposition of a couple of days

scale with zero mean and standard deviation of one. Mathematically, for time series $\boldsymbol{y}$ it can be expressed as

$$y_t^* = \frac{y_t - E(\boldsymbol{y})}{\sqrt{V(\boldsymbol{y})}}$$

where $E(\boldsymbol{y})$ is the mean and $V(\boldsymbol{y})$ is the variance of $\boldsymbol{y}$.

The point forecasts $\hat{\boldsymbol{y}}^*$ produced by neural network trained on standardised data are naturally in this new scale. In order to interpret these forecasts in a meaningful way, the output of neural network needs to be back-transformed to the original scale. Therefore it is necessary to preserve the mean $E(\boldsymbol{y})$ and variance $V(\boldsymbol{y})$ used in standardisation and compute the point forecasts in the original scale $\hat{\boldsymbol{y}}$ by performing the inverse transformation

$$\hat{y}_{t+i|t} = \hat{y}^*_{t+i|t}\sqrt{V(\boldsymbol{y})} + E(\boldsymbol{y}).$$

10.4.2 Mean adjustment

Since out aim is to forecast load profile for the entire day, it may be beneficial to remove the daily seasonality by encoding the data as deviations from the average load profile. Because of the fact that load profiles for individual weekdays can differ significantly, especially in the case of business days versus weekends (see Table 3.5), we consider separate average load profile for Sunday, Monday, Tuesday, Wednesday, Thursday, Friday and Saturday. The profile for each day of the data is mean-adjusted by subtracting the average load profile for that particular day as seen in Table 3.5.

11. Imputation

Imputation is a general term used to describe the reconstruction of missing observations from available data [86]. There are various methods suitable for data imputation. In this chapter we describe the methodology and compare various imputation methods used in this thesis.

Broadly speaking, to compare the imputation methods we artificially remove some observations, impute them using a number of methods and then evaluate the accuracy by comparing imputed data with the original data. The observations are removed according to a probability distribution that reflects the true distribution of outages in the data. Because the process of removing observations contains randomness, observations are removed and imputed 10 times and the final accuracy is averaged across all 10 runs. The implementation of imputation methods comes from R's package *imputeTS* [87].

Imputation is performed on the raw data (one observation per minute) before any other preprocessing method, namely aggregation. This is done in order to prevent aggregation from discarding valid observations in a half-hour time interval containing only a few missing minutes (see Figure 3.2). The following is a list of all imputation methods under consideration:

- random: replace missing observation by a random value between minimum and maximum of data;
- replace: replace missing observation with mean median or mode;
- moving average smoother: missing observation replacement by weighted moving average with simple, linear or exponential weighting scheme and window size $w = (1, 2, \ldots, 10)$ ($w = 2$ encompasses two past and two future observations);
- nocb: next observation carried backwards, if there is no such observation (outage at the end of time series) carry forward last observation;
- locf: last observation carried forwards, if there is no such observation (outage at the beginning of time series) carry backwards next observation;
- interpolation: uses either linear [88], spline [89] or Stineman interpolation (stine) [90] to replace missing observations;
- ARMA model;

Moreover, before imputation one can preprocess the data to possibly improve the results. In this thesis we use these preprocessing methods:

- deseasonalisation: data is decomposed and seasonal components are removed before imputation, see Section 10.4,
- splitting: data is split into seasons and each season is imputed separately.

Before we can describe the methodology of evaluating these imputation methods, we need to establish a couple of preliminary definitions. Let O be the set containing all outages present in data. The distribution of outages is then a function $d(o) : O \to \mathbb{R}$ that assigns probability of occurrence to each outage. Formally, it is defined as

$$d(o) = \frac{c(o)}{\sum_{q \in O} c(q)}$$

where $c(o) : O \to \mathbb{N}$ is a counting function assigning the total number of occurrences to each outage.

Algorithm 7 Imputation

```
1: the distribution $d$ of outages in the data $O$ is computed.
2: $\boldsymbol{y}_{t_1,t_2} \leftarrow$ largest subset of time series $\boldsymbol{y}$ without any outage
3: for $i = 1, 2, \dots, 10$ do
4:     $\boldsymbol{y}^{out}_{t_1,t_2} \leftarrow \boldsymbol{y}_{t_1,t_2}$
5:     $j \leftarrow t_1$
6:     while $j \leq t_2$ do                                   ▷ adding outages
7:         draw outage $o$ from the distribution $d$           ▷ roulette wheel selection
8:         if $o > 0$ then
9:             $\boldsymbol{y}^{out}_{j,j+o-1} = \mathbf{0}_o$     ▷ $\mathbf{0}_o$ is vector of $o$ zeros
10:            $j \leftarrow j + o$
11:        else
12:            $j \leftarrow j + 1$
13:        end if
14:    end while
15:    for each imputation method $\boldsymbol{m}$ : data $\to$ imputed data do
16:        $\boldsymbol{y}^{imp}_{t_1,t_2} \leftarrow \boldsymbol{m}(\boldsymbol{y}^{out}_{t_1,t_2})$
17:        evaluate performance by comparing $\boldsymbol{y}^{imp}_{t_1,t_2}$ to $\boldsymbol{y}_{t_1,t_2}$
18:    end for
19: end for
20: average the performances across all 10 runs
```

The whole process of removing observations, imputation and computing the accuracy is summarised in Algorithm 7.

Since various methods use very different parameters, for clarity we chose to list experimental results in three separate tables. Table 11.1 describes the performance of moving average smoother, Table 11.2 contains the results of interpolation and Table 11.3 displays the results of all the other methods, including random, replace, nocb, locf and ARMA model.

Table 11.1: Imputation results for moving average smoother

prep	weighting scheme	window size	SRMSE	MASE	SMAPE	SMAE
none	exponential	2	0.1980	0.4218	0.3353	0.2953
none	linear	2	0.1985	0.4239	0.3376	0.2940
none	simple	2	0.1988	0.4250	0.3386	0.2941
none	exponential	1	0.1996	0.4208	0.3343	0.3027
none	linear	1	0.2000	0.4225	0.3363	0.3026

Continued on the next page

Table 11.1 – continued from the previous page

prep	weighting scheme	window size	SRMSE	MASE	SMAPE	SMAE
none	simple	1	0.2001	0.4230	0.3367	0.3026
des	simple	2	0.2043	0.4218	0.3057	0.3062
none	exponential	3	0.2061	0.4274	0.3373	0.3158
des	linear	2	0.2068	0.4236	0.3066	0.3118
des	exponential	2	0.2085	0.4245	0.3075	0.3154
none	exponential	4	0.2104	0.4310	0.3386	0.3274
none	exponential	5	0.2117	0.4328	0.3397	0.3303
none	linear	3	0.2119	0.4329	0.3407	0.3310
none	exponential	6	0.2123	0.4341	0.3406	0.3310
des	exponential	3	0.2123	0.4272	0.3080	0.3303
none	exponential	7	0.2126	0.4350	0.3415	0.3306
none	exponential	8	0.2129	0.4359	0.3422	0.3315
none	exponential	9	0.2133	0.4367	0.3430	0.3320
des	exponential	5	0.2133	0.4284	0.3086	0.3338
des	exponential	6	0.2133	0.4294	0.3097	0.3328
des	exponential	7	0.2133	0.4305	0.3112	0.3316
des	exponential	4	0.2134	0.4281	0.3082	0.3345
des	exponential	8	0.2136	0.4319	0.3127	0.3314
none	exponential	10	0.2136	0.4376	0.3437	0.3322
des	exponential	9	0.2138	0.4330	0.3139	0.3312
des	exponential	10	0.2140	0.4339	0.3151	0.3312
des	linear	3	0.2141	0.4282	0.3077	0.3368
des	simple	3	0.2165	0.4304	0.3090	0.3441
des	linear	4	0.2176	0.4317	0.3099	0.3458
none	simple	3	0.2183	0.4377	0.3429	0.3488
des	linear	5	0.2185	0.4352	0.3141	0.3433
des	exponential	1	0.2192	0.4318	0.3144	0.3449
des	linear	1	0.2194	0.4323	0.3144	0.3451
des	simple	1	0.2195	0.4326	0.3146	0.3451
des	linear	6	0.2196	0.4404	0.3192	0.3401
des	linear	7	0.2197	0.4456	0.3243	0.3364
none	linear	4	0.2212	0.4405	0.3435	0.3557
des	linear	8	0.2218	0.4528	0.3300	0.3379
des	simple	4	0.2232	0.4362	0.3127	0.3590
des	linear	9	0.2235	0.4588	0.3342	0.3382
des	simple	5	0.2251	0.4453	0.3194	0.3571
none	linear	5	0.2252	0.4457	0.3463	0.3637
des	linear	10	0.2255	0.4648	0.3391	0.3387
des	simple	6	0.2275	0.4530	0.3263	0.3567
none	linear	6	0.2276	0.4502	0.3495	0.3685
des	simple	7	0.2282	0.4591	0.3324	0.3471
none	linear	7	0.2286	0.4542	0.3528	0.3678
none	linear	8	0.2311	0.4597	0.3566	0.3739
none	simple	4	0.2319	0.4489	0.3471	0.3840

Continued on the next page

Table 11.1 – continued from the previous page

prep	weighting scheme	window size	SRMSE	MASE	SMAPE	SMAE
des	simple	8	0.2319	0.4689	0.3395	0.3562
none	linear	9	0.2338	0.4666	0.3614	0.3799
des	simple	9	0.2347	0.4774	0.3450	0.3611
none	linear	10	0.2368	0.4722	0.3653	0.3849
none	simple	5	0.2371	0.4559	0.3509	0.3975
des	simple	10	0.2377	0.4844	0.3505	0.3666
none	simple	6	0.2404	0.4619	0.3551	0.4009
none	simple	7	0.2413	0.4666	0.3593	0.4045
none	simple	8	0.2452	0.4754	0.3651	0.4120
none	simple	9	0.2492	0.4855	0.3721	0.4170
none	simple	10	0.2537	0.4951	0.3782	0.4218
split	simple	4	0.4453	0.9619	0.6288	0.7294
split	linear	4	0.4505	0.9256	0.5902	0.7727
split	exponential	4	0.4657	0.9043	0.5578	0.8326
split	simple	3	0.4671	0.9724	0.6178	0.7961
split	linear	3	0.4679	0.9273	0.5789	0.8296
split	exponential	5	0.4727	0.9384	0.5810	0.8408
split	exponential	3	0.4751	0.9087	0.5543	0.8588
split	exponential	10	0.4766	0.9859	0.6271	0.8340
split	exponential	8	0.4766	0.9848	0.6259	0.8347
split	exponential	9	0.4766	0.9861	0.6271	0.8343
split	exponential	7	0.4777	0.9879	0.6275	0.8374
split	exponential	6	0.4799	0.9927	0.6299	0.8422
split	linear	5	0.4829	1.0395	0.6571	0.8094
split	linear	10	0.5012	1.2166	0.7996	0.7342
split	exponential	2	0.5028	0.9452	0.5509	0.9347
split	linear	8	0.5028	1.2106	0.7946	0.7496
split	linear	9	0.5058	1.2381	0.8105	0.7415
split	linear	2	0.5090	0.9664	0.5624	0.9448
split	simple	5	0.5100	1.1502	0.7206	0.8014
split	linear	7	0.5198	1.2739	0.8256	0.7792
split	simple	2	0.5237	1.0141	0.5848	0.9601
split	simple	1	0.5244	0.9630	0.5514	0.9412
split	linear	1	0.5244	0.9630	0.5514	0.9412
split	exponential	1	0.5244	0.9630	0.5514	0.9412
split	simple	10	0.5299	1.3460	0.8640	0.7414
split	simple	8	0.5406	1.3729	0.8769	0.7339
split	simple	9	0.5443	1.4144	0.8955	0.7202
split	linear	6	0.5446	1.3449	0.8580	0.8235
split	simple	7	0.5785	1.5134	0.9363	0.7664
split	simple	6	0.6408	1.7298	1.0212	0.8412

Continued on the next page

Table 11.1 – continued from the previous page

prep	weighting scheme	window size	SRMSE	MASE	SMAPE	SMAE

"none" means that no preprocessing method is used.
"split" indicates that data was split into seasons and each season was imputed separately.
"des" indicates that deseasonalisation before imputation was performed.
Results are sorted lexicographically in ascending order according to SRMSE.

Table 11.2: Imputation results for interpolation

prep	type	SRMSE	MASE	SMAPE	SMAE
none	stine	0.2166	0.5102	0.3751	0.3186
none	linear	0.2219	0.5195	0.3791	0.3233
des	stine	0.2240	0.5223	0.4026	0.3409
none	spline	0.2266	0.4946	0.3497	0.3648
des	linear	0.2402	0.5720	0.4148	0.3573
des	spline	0.2451	0.5308	0.4102	0.4041
split	linear	0.5244	0.9630	0.5514	0.9412
split	stine	0.5302	0.9690	0.5530	0.9546
split	spline	0.7545	1.5040	1.0177	1.2765

"none" means that no preprocessing method is used.
"split" indicates that data was split into seasons and each season was imputed separately.
"des" indicates that deseasonalisation before imputation was performed.
Results are sorted in ascending order by SRMSE.

It is clear that moving average smoother and its modifications performed best. For imputation of the whole dataset we chose moving average with linear weighting scheme, window size $w = 2$ and no preprocessing as it achieved the best SMAE.

Table 11.3: Imputation results for various methods

prep	alg	SRMSE	MASE	SMAPE	SMAE
none	arma	0.2443	0.9230	0.8868	0.3001
none	locf	0.2753	0.8155	0.6075	0.3324
des	locf	0.2907	0.8524	0.6200	0.3626
des	arma	0.2976	1.4506	1.3672	0.3113
none	nocb	0.3393	0.9693	0.8234	0.4073
des	nocb	0.3485	0.9480	0.7705	0.4299
split	median	0.3958	0.7492	0.5117	0.7468
des	median	0.4217	0.9011	0.6336	0.7182
des	mean	0.4802	1.2309	0.8240	0.6572
split	mean	0.5145	1.4425	0.9006	0.6664
split	mode	0.5582	1.0401	0.8317	1.0926
none	median	0.6048	1.3904	0.8956	0.9532
split	nocb	0.6397	1.2880	0.9177	1.1287
none	mode	0.6688	1.1425	0.8193	1.2856
split	locf	0.8674	1.8790	0.8635	1.2949
none	mean	0.9025	2.8217	1.3847	0.9508
des	mode	0.9033	2.1453	3.1129	1.5067
split	arma	1.1717	3.2721	4.9460	2.4941
split	random	2.5060	7.5192	1.7568	2.9321
none	random	6.2046	21.0529	2.5897	4.6809
des	random	9.5344	37.8325	2.8098	5.5562

”none” means that no preprocessing method is used.
”split” indicates that data was split into seasons and each season was imputed separately.
”des” indicates that deseasonalisation before imputation was performed.
”locf” stands for last observation carried forward.
”nocb” stands for next observation carried backwards.
”mean”, ”median”, ”mode”, ”random” describes what value is used to replace the missing observations.
Results are sorted in ascending order according to SRMSE.

12. Experiments

In this chapter we present the results of conducted experiments and compare the performace of various forecasting models. The implementation of ES, ARMA and ARMAX models comes from R's package `forecast` [91]. Neural networks were implemented using python's `Keras` library [92].

For convenience when we refer to the best or most accurate model we do it according to SRMSE.

12.1 ES models

For the evaluation of ES models both weekly and monthly cross-validation was used (Section 10.2). At each step of the cross-validation procedure, the training process described in Section 6.3 took place. Each of the thirty ES models was initialised, estimated and compared using AICc and the best one was selected to forecast next week or month by alternating between forecasting and filtering (Section 5.5).

To account for multiple seasonalities, the training and evaluation of ES models was performed also on week-adjusted data, hour-adjusted data and their combination (week adjustment first). For non-adjusted data the assumed frequency of time series was $m = 48$ (daily), for hour-adjusted data it was $m = 7$ (weekly), for week-adjusted data it was $m = 48$ (daily) and for the combination of hour and week adjustment it was $m = 52$ (annual). Other forms of preprocessing that were considered include the Box-Cox transformation, deseasonalisation and their combination (deseasonalisation first).

The corresponding experimental results for monthly cross-validation found in Table 12.1 indicate a clear dominance of forecasts on hour-adjusted data. Furthermore it is clear that week adjustment did not generally improve the performance. This may be caused by the fact that the training set for week-adjusted models is only 1/7 of the original train set, which may be too small a training set especially when data is also hour-adjusted. Additionally, deseasonalisation proved to be detrimental to the performance of time-adjusted models which is to be expected as both techniques are designed to deal with seasonality. Box-Cox transformation worsens SRMSE and SMAE of some models while improving their MASE and SMAPE.

We can conclude that hour adjustement is the most efficient preprocessing method for ES models. The worst preprocessing method seems to be the week adjustment.

In Figure 12.1 we compare the best and worst days for the best ES model.

Experimental results for weekly cross-validation are listed in Table 12.2. Figure 12.2 summarises the best and worst days for the best ES model for weekly cross-validation.

Again the best performance is achieved on hour-adjusted data. Deseasonalisation is not suitable for ES models and weekly cross-validation as it worsened the performance in almost all cases. Similarly to deseasonalisation, in most cases Box-Cox transformation is also not beneficial. The worst preprocessing methods are again week adjustment.

Table 12.1: Experimental results for ES models using monthly cross-validation

wa	ha	des	bc	SRMSE	MASE	SMAPE	SMAE
✗	✓	✗	✗	0.6390	0.7723	0.2462	4.3332
✗	✓	✓	✗	0.6456	0.7678	0.2444	4.6052
✗	✓	✗	✓	0.6582	0.7408	0.2282	4.6735
✗	✓	✓	✓	0.6637	0.7497	0.2303	5.0512
✓	✓	✗	✗	0.6744	0.8172	0.2547	4.5755
✓	✓	✗	✓	0.6934	0.7939	0.2448	4.8563
✓	✓	✓	✗	0.7942	0.9157	0.2669	5.1179
✗	✗	✓	✗	0.8374	1.0199	0.3316	5.1632
✓	✗	✓	✗	0.8659	1.0662	0.3573	5.2755
✗	✗	✗	✗	0.9469	1.1276	0.3412	5.3710
✗	✗	✗	✓	0.9506	1.1298	0.3435	5.4291
✓	✓	✓	✓	0.9534	0.9570	0.2741	50.6483
✓	✗	✗	✓	0.9735	1.1918	0.3645	5.7481
✓	✗	✓	✓	1.3051	1.2851	0.3224	19.0228
✗	✗	✓	✓	1.4145	1.2648	0.3093	23.7938
✓	✗	✗	✗	1.8406	2.4646	0.4514	6.4907

"wa", "ha", "bc" and "des" indicate the type of preprocessing used (week adjustement, hour adjustement, Box-Cox transformation, deseasonalisation).
Results are sorted in ascending order first by SRMSE.

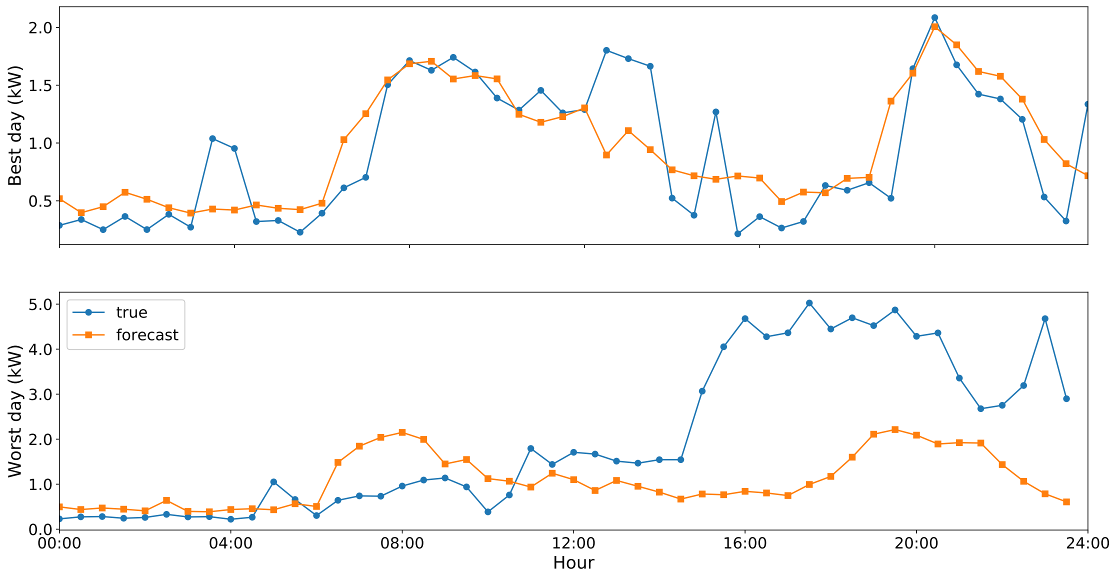

Figure 12.1: Best and worst load profile forecasts of the best ES model when using monthly cross-validation

We conclude that hour adjustment is the most efficient preprocessing method for ES models. Also monthly cross-validation as a whole is generally better

Table 12.2: Experimental results for ES models using weekly cross-validation

wa	ha	des	bc	SRMSE	MASE	SMAPE	SMAE
✗	✓	✗	✗	0.6393	0.7707	0.2458	4.3310
✗	✓	✗	✓	0.6478	0.7349	0.2282	4.5101
✓	✓	✗	✗	0.6754	0.8165	0.2543	4.5757
✓	✓	✗	✓	0.6837	0.7876	0.2426	4.6751
✗	✓	✓	✗	0.7066	0.8319	0.2584	4.6026
✗	✓	✓	✓	0.7120	0.8101	0.2468	5.4737
✓	✓	✓	✗	0.7950	0.9171	0.2674	5.2135
✗	✗	✓	✗	0.8557	1.0123	0.3179	5.1655
✓	✗	✓	✗	0.8710	1.0784	0.3610	5.2621
✓	✓	✓	✓	0.9139	0.9600	0.2747	31.2254
✗	✗	✗	✗	0.9414	1.1175	0.3382	5.3708
✗	✗	✗	✓	0.9476	1.1231	0.3415	5.4393
✓	✗	✗	✓	0.9749	1.1927	0.3664	5.7495
✓	✗	✓	✓	1.3507	1.3189	0.3276	19.0681
✗	✗	✓	✓	1.6694	1.3911	0.3058	23.7651
✓	✗	✗	✗	1.8520	2.4860	0.4532	6.4301

"wa", "ha", "bc" and "des" indicate the type of preprocessing used (week adjustement, hour adjustement, Box-Cox transformation, deseasonalisation).
Results are sorted lexicographically in ascending order first by SRMSE, then MASE, SMAPE and finally by SMAE.

suited for ES models. Weekly cross-validation has either negligible or significantly detrimental effects on performance. However, the top performing models from both monthly and weekly cross-validation are comparable in their performance. Some models have significantly larger SMAE values caused by a small number of forecasts significantly larger that the actual load. However, after examining each case we were unable to find overarching cause of this phenomenon.

12.2 ARMA models

ARMA models were evaluated using monthly cross-validation (Section 10.2). At each step of the cross-validation procedure, the models were trained as described in Section 7.3 and the best one according to AICc was selected to forecast the next month by alternating between forecasting and filtering (Section 5.5).

Preprocessing methods considered for ARMA models were the same as for ES models, i.e. week adjustement, hour adjustement and their combination (week adjustement first) in addition to Box-Cox transformation and deseasonalisation (deseasonalisation first). The assumed frequency of the time series was $m = 48$ (daily) for unadjusted and week-adjusted data, $m = 7$ (weekly) for hour-adjusted data and $m = 52$ (annual) for the combination of hour and week adjustment.

The results, listed in Table 12.3, again hint at the domination of hour adjustment over week adjustement and no adjustement. On the other hand, week

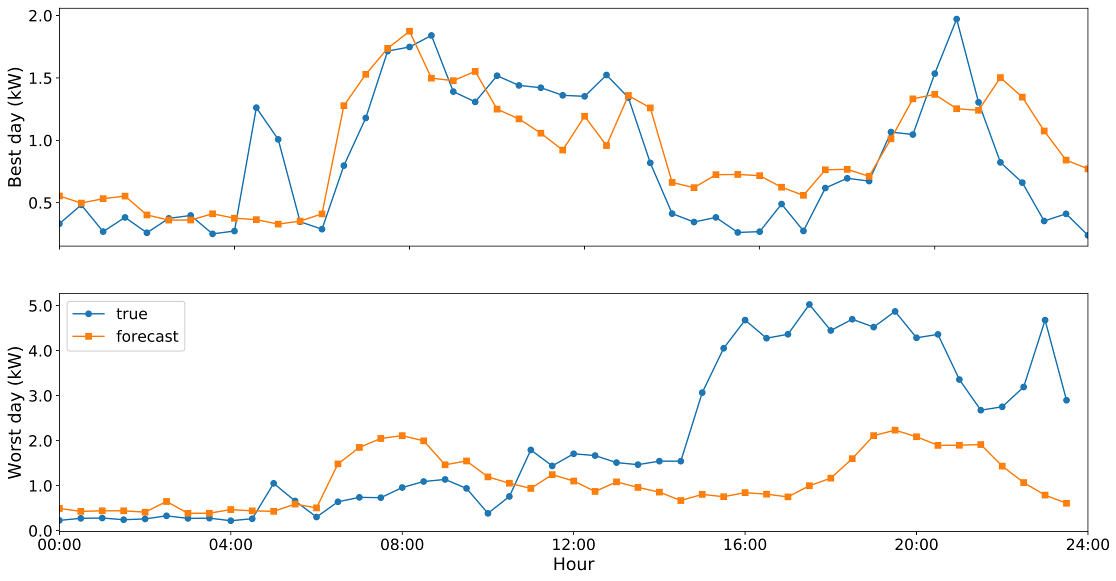

Figure 12.2: Best and worst load profile forecasts of the best ES model when using weekly cross-validation

Table 12.3: Experimental results for ARMA models

wa	ha	des	bc	SRMSE	MASE	SMAPE	SMAE
✗	✓	✓	✗	0.6321	0.7589	0.2429	4.4170
✗	✓	✗	✗	0.6406	0.7758	0.2479	4.4284
✗	✓	✓	✓	0.6641	0.7440	0.2291	4.7579
✓	✓	✗	✗	0.6663	0.8074	0.2515	4.5729
✗	✗	✓	✗	0.6724	0.8393	0.2853	4.4524
✗	✓	✗	✓	0.6728	0.7543	0.2327	4.9779
✗	✗	✗	✓	0.6747	0.8415	0.2846	4.4515
✓	✗	✓	✗	0.6842	0.8606	0.2806	4.5731
✗	✗	✓	✓	0.6871	0.7886	0.2430	4.7399
✓	✗	✓	✓	0.6905	0.8058	0.2501	4.7860
✓	✓	✗	✓	0.6956	0.7924	0.2445	4.7393
✗	✓	✓	✗	0.7067	0.8335	0.2587	4.6872
✓	✓	✓	✗	0.7920	0.9185	0.2703	5.4253
✓	✗	✗	✗	0.8305	1.0725	0.3333	5.2270
✗	✗	✗	✗	0.8313	1.0401	0.3124	5.3813
✓	✗	✗	✓	0.8733	1.0733	0.3302	5.4572
✓	✓	✓	✓	12.0091	1.2235	0.2789	1388.3484

"wa", "ha", "bc" and "des" indicate the type of preprocessing used (week adjustement, hour adjustement, Box-Cox transformation, deseasonalisation).
Results are sorted in ascending order according to SRMSE.

adjustment resulted in decreased performance in all cases. Deseasonalisation improves the performance in most cases. Box-Cox transformation increases SRMSE and SMAE, but can help to lower MASE and SMAPE.

Out of all the preprocessing methods under consideration, hour adjustment appears to be the best, while week adjustment seems to be the worst. We were unable to determine the cause of unusually high SMAE of one of the models. After we replaced only the largest residual with zero, the performance of the model improved significantly (see Table 12.4).

Table 12.4: The performance of the worst ARMA model with the largest residual replaced by zero

wa	ha	des	bc	SRMSE	MASE	SMAPE	SMAE
✓	✓	✓	✓	1.1391	1.0033	0.2783	41.7385

"wa", "ha", "bc" and "des" indicate the type of preprocessing used (week adjustement, hour adjustement, Box-Cox transformation, deseasonalisation).
Results are sorted in ascending order according to SRMSE.

Figure 12.3 contains a comparison between the best and worst day for the best performing ARMA model.

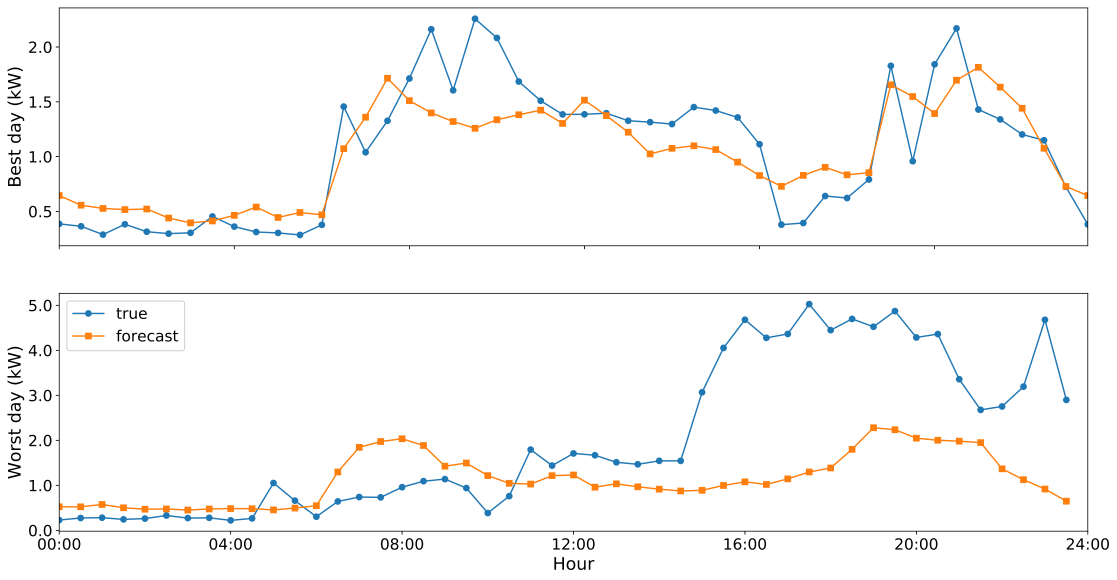

Figure 12.3: Best and worst load profile forecasts of the best ARMA model

12.3 ARMAX models

Monthly cross-validation was used for evaluation of ARMAX models (Section 10.2). At each step of the cross-validation procedure, the models were trained as described in Section 8.4. The best model (according to AICc) was chosen to forecast the next month by alternating between forecasting and filtering (Section 5.5).

Preprocessing methods considered for ARMAX models were the same as for ES and ARMA models, i.e. week adjustement, hour adjustement and their combination (week adjustement first) together with Box-Cox transformation and deseasonalisation. Moreover, for ARMAX models we also considered two types of regression. Some models used weather regression on the weather data described in Section 3.3, some models used Fourier regression described in Section 8.1.1 and some models used both.

In weather regression every weather characteristic (temperature, humidity, wind speed) was standardised (see Section 10.4.1) before training the model. When using only weather regression, for unadjusted data the assumed frequency of time series was $m = 48$ (daily), for hour-adjusted data it was $m = 7$ (weekly), for week-adjusted data it was $m = 48$ (daily) and for the combination of hour and week adjustment it was $m = 52$ (annual).

When using Fourier regression, only unadjusted and hour-adjusted time adjustments are considered. We let ARMAX models handle one type of seasonality (annual, i.e. $m = 365 \cdot 48$ for unadjusted data and $m = 365$) for hour-adjusted data. Fourier regression is used to model all other seasonalities. Fourier orders are $k_1 = 3$ and $k_2 = 5$ (modelling daily and weekly seasonality for non adjusted data) and $k = 3$ (modelling weekly seasonality for hour-adjusted data). It may be possible to archive better performance using orders found by a grid search or hill climbing algorithms, but the computation time is high as it is and therefore we did not include this type of parameter tuning in this thesis. Fine-tuning Fourier orders may the subject of future work.

Table 12.5: Experimental results for ARMAX models

wa	ha	des	bc	wregs	fregs	SRMSE	MASE	SMAPE	SMAE
✗	✓	✗	✗	✗	✓	0.6384	0.7673	0.2434	4.3621
✗	✓	✗	✗	✓	✓	0.6392	0.7729	0.2466	4.4429
✗	✓	✗	✗	✓	✗	0.6468	0.7886	0.2516	4.5711
✗	✓	✗	✓	✓	✓	0.6502	0.7408	0.2299	4.6432
✗	✓	✓	✗	✓	✗	0.6548	0.8043	0.2553	4.5740
✗	✓	✗	✓	✓	✗	0.6661	0.7604	0.2354	4.6950
✓	✓	✗	✗	✓	✗	0.6663	0.8101	0.2541	4.3666
✗	✓	✗	✓	✗	✓	0.6676	0.7499	0.2311	4.7450
✓	✗	✓	✗	✓	✗	0.6744	0.8484	0.2740	4.5247
✗	✓	✓	✓	✓	✗	0.6766	0.7837	0.2431	4.7181
✗	✗	✗	✗	✓	✓	0.6777	0.8506	0.2952	4.3372
✗	✗	✗	✗	✗	✓	0.6780	0.8528	0.2970	4.3485
✗	✗	✓	✗	✓	✗	0.6844	0.8630	0.2776	4.6997
✗	✗	✗	✓	✓	✓	0.6936	0.8103	0.2525	4.7557
✓	✓	✗	✓	✓	✗	0.6987	0.7974	0.2458	4.8171
✗	✗	✗	✓	✗	✓	0.7035	0.8157	0.2523	4.8015
✓	✗	✓	✓	✓	✗	0.7101	0.8223	0.2555	4.8975
✗	✗	✓	✓	✓	✗	0.7155	0.8317	0.2583	4.9606
✗	✗	✗	✗	✓	✗	0.7454	0.9716	0.3047	4.9304

Continued on the next page

Table 12.5 – continued from the previous page

wa	ha	des	bc	wregs	fregs	SRMSE	MASE	SMAPE	SMAE
✗	✗	✗	✓	✓	✗	0.7811	0.9620	0.3037	5.1691
✓	✓	✓	✗	✓	✗	0.7916	0.9151	0.2689	5.4827
✓	✗	✗	✗	✓	✗	0.7998	1.0504	0.3276	4.9414
✓	✗	✗	✓	✓	✗	0.8262	1.0212	0.3267	5.4136
✗	✓	✓	✓	✗	✓	0.8305	0.9744	0.3007	5.8806
✗	✓	✓	✗	✓	✓	0.8727	1.0674	0.3460	5.0889
✗	✗	✓	✗	✗	✓	0.8745	1.1075	0.3500	4.8992
✗	✗	✓	✗	✓	✓	0.8774	1.1106	0.3527	4.8280
✗	✗	✓	✓	✓	✓	0.8900	1.0555	0.3209	7.5615
✗	✓	✓	✗	✗	✓	0.8973	1.0992	0.3429	4.9298
✗	✓	✓	✗	✗	✓	0.9139	1.1177	0.3460	5.3586
✗	✗	✓	✓	✗	✓	0.9219	1.0682	0.3197	9.6521
✓	✓	✓	✓	✓	✗	0.9294	0.9389	0.2759	34.7992
✗	✓	✓	✓	✓	✓	8.8280	1.1501	0.2993	1136.3982

"wa", "ha", "bc" and "des" indicate the type of preprocessing used (week adjustement, hour adjustement, Box-Cox transformation, deseasonalisation).
"wregs" (weather regression) and "fregs" (Fourier regression) describe the type of regression.
Results are sorted in ascending order according to SRMSE.

From table 12.5 it is clear that hour adjustment is a better preprocessing method compared to week adjustment or no time adjustment, as the top of the table is dominated by models using it. Unsurprisingly, the combination of Fourier regression and deseasonalisation performed the worst as both target the same aspect of data. However, deseasonalisation can help weather regression, especially when no time adjustment is used. Box-Cox transformation usually has detrimental effects on the performance according to SRMSE and SMAE, but can improve MASE and SMAPE. Fourier regression is overall better than weather regression. This is especially apparent when comparing the performance of models trained on unadjusted data.

From the results we can conclude that the best preprocessing method is again hour-adjustment. Deseasonalisation seems to be the worst. However, deseasonalisation is overall better suited for modelling multiple seasonalities, as ARMA models trained on deseasonalised data performed better than their Fourier-regression-using-ARMAX counterparts. We were unable to determine the overarching cause of unusually high SMAE of several of the models. After we replaced only the largest residual of the worst model with zero, the performance improved significantly (see Table 12.6).

In Figure 12.4 we compare the best and worst forecasts of the best performing ARMAX model.

12.4 Neural networks

To construct the training samples for neural networks, we initially set window shift $s = 48$ (see Section 9.4) since forecast horizon is $h = 48$. However, this method resulted in poor performance as the number of training samples was approximately

Table 12.6: The performance of the worst ARMAX model with the largest residual replaced by zero

wa	ha	des	bc	wregs	fregs	SRMSE	MASE	SMAPE	SMAE
✗	✓	✓	✓	✓	✓	2.2412	1.0425	0.2993	195.0521

"wa", "ha", "bc" and "des" indicate the type of preprocessing used (week adjustement, hour adjustement, Box-Cox transformation, deseasonalisation).
Results are sorted in ascending order according to SRMSE.

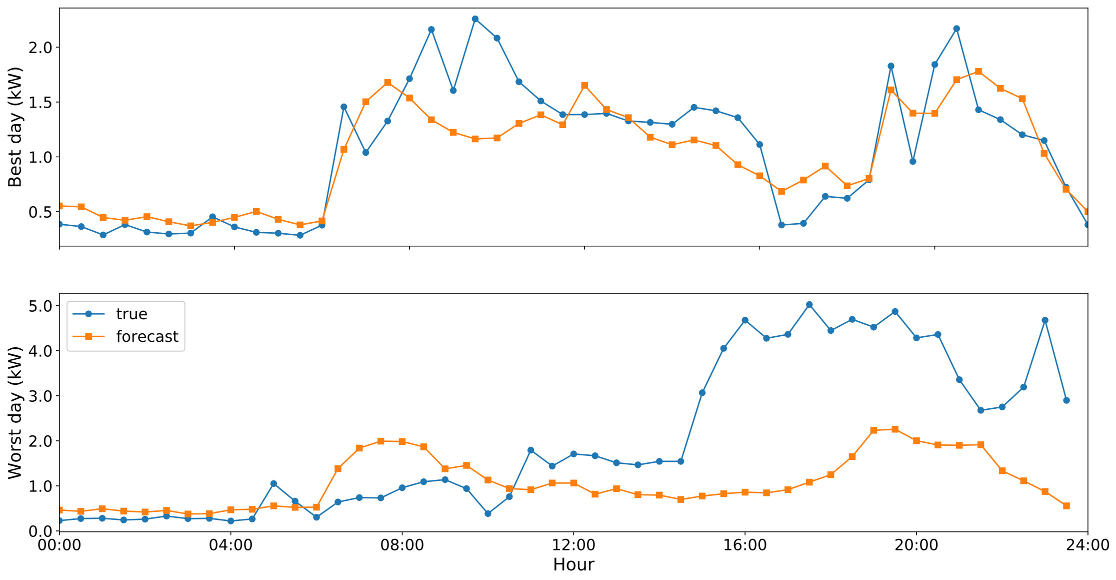

Figure 12.4: Best and worst load profile forecasts of ARMAX with Fourier regression trained on hourly adjusted data

the same as the number of days in the training set, which is only 1071 (see Section 10.2). We therefore set $s = 1$, which increased the number of training samples to approximately $48 \cdot 1071$ which improved the performance.

Table 12.7: Experimental results for SLNs

prep	weather	w	hid	activation	SRMSE	MASE	SMAPE	SMAE
mean	✗	1,7	100	sigmoid	0.6348	0.7696	0.2545	4.4832
mean	✗	1,7	50	sigmoid	0.6358	0.7718	0.2573	4.4298
mean	✗	1	100	sigmoid	0.6376	0.7787	0.2595	4.2843
mean	✗	1	100	tanh	0.6395	0.7815	0.2622	4.2307
mean	✗	1	50	sigmoid	0.6398	0.7847	0.2657	4.2790
mean	✗	1,7	200	sigmoid	0.6408	0.7830	0.2627	4.4108
mean	✗	1	200	sigmoid	0.6414	0.7879	0.2632	4.2072
mean	✗	1	50	tanh	0.6417	0.7831	0.2608	4.3350
mean	✗	1	200	tanh	0.6427	0.7869	0.2639	4.4392

Continued on the next page

Table 12.7 – continued from the previous page

prep	weather	w	hid	activation	SRMSE	MASE	SMAPE	SMAE
mean	✗	1,7	50	tanh	0.6433	0.7839	0.2644	4.5692
mean	✓	1	50	sigmoid	0.6446	0.7890	0.2640	4.2181
mean	✓	1	100	sigmoid	0.6447	0.7859	0.2626	4.2130
mean	✗	1,7	100	tanh	0.6452	0.7875	0.2605	4.5314
mean	✓	1,7	50	sigmoid	0.6453	0.7873	0.2624	4.3539
mean	✓	1,7	100	sigmoid	0.6477	0.7869	0.2613	4.3430
mean	✓	1,7	50	tanh	0.6488	0.7975	0.2693	4.3556
mean	✓	1	50	tanh	0.6501	0.7951	0.2701	4.2708
mean	✓	1	200	sigmoid	0.6517	0.8014	0.2674	4.1938
none	✗	1,7	100	sigmoid	0.6553	0.7962	0.2556	4.7819
mean	✗	1,7	200	tanh	0.6570	0.8025	0.2717	4.6438
mean	✓	1,7	200	sigmoid	0.6583	0.8094	0.2763	4.2642
none	✓	1,7	50	tanh	0.6584	0.8074	0.2646	4.8589
none	✓	1,7	50	sigmoid	0.6588	0.8122	0.2669	4.7741
none	✗	1,7	50	tanh	0.6598	0.8094	0.2618	4.8788
mean	✓	1	100	tanh	0.6606	0.8089	0.2767	4.2855
none	✗	1,7	50	sigmoid	0.6610	0.8049	0.2620	4.9869
none	✓	1,7	100	sigmoid	0.6626	0.8159	0.2711	4.6756
mean	✓	1	200	tanh	0.6629	0.8199	0.2809	4.2040
none	✗	1,7	200	sigmoid	0.6660	0.8112	0.2647	5.0397
mean	✓	1,7	100	tanh	0.6669	0.8239	0.2822	4.3349
none	✗	1,7	100	tanh	0.6673	0.8058	0.2644	5.3518
none	✓	1	50	sigmoid	0.6674	0.8181	0.2698	4.4190
none	✓	1	50	tanh	0.6683	0.8215	0.2712	4.4114
none	✗	1	100	sigmoid	0.6694	0.8187	0.2669	4.7085
none	✓	1	100	sigmoid	0.6695	0.8239	0.2777	4.6780
none	✗	1	50	sigmoid	0.6701	0.8250	0.2643	4.8626
mean	✓	1,7	200	tanh	0.6709	0.8335	0.2859	4.1370
none	✓	1,7	200	sigmoid	0.6710	0.8286	0.2808	4.2938
none	✗	1	50	tanh	0.6715	0.8285	0.2678	4.7070
none	✗	1	200	sigmoid	0.6724	0.8255	0.2687	4.8945
none	✗	1	200	tanh	0.6736	0.8274	0.2717	4.7716
none	✗	1	100	tanh	0.6750	0.8288	0.2713	4.7180
none	✗	1,7	200	tanh	0.6778	0.8308	0.2765	4.8253
none	✓	1	200	sigmoid	0.6790	0.8289	0.2810	4.5878
none	✓	1,7	100	tanh	0.6797	0.8388	0.2822	4.6799
none	✓	1	100	tanh	0.6801	0.8408	0.2886	4.4990
none	✓	1	200	tanh	0.6919	0.8622	0.2932	4.5562
none	✓	1,7	200	tanh	0.6929	0.8557	0.2945	4.7197
des	✗	1	50	sigmoid	0.7288	0.8924	0.2902	4.5845
des	✗	1	100	sigmoid	0.7306	0.8980	0.2947	4.4500
des	✗	1,7	50	sigmoid	0.7319	0.8998	0.2982	4.6505
des	✗	1,7	100	sigmoid	0.7321	0.8993	0.2984	4.4988
des	✗	1,7	50	tanh	0.7330	0.9036	0.2998	4.7019

Continued on the next page

Table 12.7 – continued from the previous page

prep	weather	w	hid	activation	SRMSE	MASE	SMAPE	SMAE
des	✗	1	200	sigmoid	0.7332	0.9041	0.2956	4.4368
des	✗	1,7	200	sigmoid	0.7342	0.9078	0.3070	4.5442
des	✗	1	100	tanh	0.7355	0.9086	0.2985	4.5824
des	✗	1	50	tanh	0.7357	0.9019	0.2950	4.4847
des	✗	1,7	100	tanh	0.7381	0.9072	0.3019	4.6118
des	✓	1,7	50	sigmoid	0.7427	0.9191	0.3105	4.4442
des	✗	1	200	tanh	0.7432	0.9172	0.3042	4.4780
des	✓	1	50	sigmoid	0.7439	0.9208	0.3092	4.6176
des	✓	1,7	100	sigmoid	0.7462	0.9248	0.3138	4.5552
des	✗	1,7	200	tanh	0.7465	0.9205	0.3070	4.7226
des	✓	1	100	sigmoid	0.7469	0.9287	0.3152	4.4584
des	✓	1	50	tanh	0.7502	0.9312	0.3103	4.5211
des	✓	1	200	sigmoid	0.7504	0.9345	0.3189	4.6398
des	✓	1,7	50	tanh	0.7508	0.9315	0.3196	4.8332
des	✓	1,7	200	sigmoid	0.7551	0.9395	0.3248	4.4659
des	✓	1	100	tanh	0.7590	0.9467	0.3229	4.5399
des	✓	1,7	100	tanh	0.7609	0.9387	0.3218	4.6055
des	✓	1	200	tanh	0.7655	0.9571	0.3274	4.7573
des	✓	1,7	200	tanh	0.7758	0.9722	0.3342	4.6376

"none", "mean" and "des" indicate the type of preprocessing used (no preprocessing, mean adjustment, deseasonalisation.
"we" marks whether weather data was included for training purposes.
w indicate the window size, 1 means the previous day, 1,7 means the previous day and the corresponding day from previous week.
"hidden" indicate the number of neurons in the hidden layer.
"activation" indicate the activation function of neurons in the hidden layer.
Results are sorted in ascending order by SRMSE.

Energy consumption data used for neural networks is always standardised (see Section 10.4.1) even when we say that no preprocessing method was used. Apart from standardisation, additional preprocessing methods considered for neural networks were two: deseasonalisation and mean adjustment. Standardisation is always performed the last, after all other preprocessing methods.

In the training process of some neural networks we also included standardised weather data and dummy variables to encode the date (see Section 9.4).

We consider preprocessing methods and the presence of weather data as a type of hyperparameters to tune. Other hyperparameters include number of layers, number of neurons in each layer, activation functions for the layers and window size (see Section 9.4). We consider one or two hidden layers, 50,100 or 200 neurons in a layer, sigmoid or tanh activation functions and window size of 1 (72 previous half-hours to capture daily seasonality, i.e. $\boldsymbol{y}_{t-72+1:t}$) or 1,7 (concatenated $\boldsymbol{y}_{t-72+1:t}$ and $\boldsymbol{y}_{t-6\times48-72+1:t-6\times48}$ for both daily and weekly seasonality). The activation function for the output layer is linear, as the energy consumption is not bounded. The minibatch size is set to 50 samples.

These hyperparameters were optimised using grid search and monthly cross-validation (Section 10.2). To combat overfitting we used the early stopping (see 9.3.4) with $k = 5$. At each step of cross-validation procedure, 20% of training samples were set aside as a validation set. This can be done because MLN are not sensitive to the order in which the training samples are presented. Also at each step of cross-validation procedure the network is initialised and trained multiple times to reduce the impact or random weights initialisation (see Section 9.3.5). For SLNs this number is 10, while MLN with two hidden layers use only three reinitialisations due to time constraints.

Table 12.8: Experimental results for MLN with two hidden layers

prep	W	w	hid 1	hid 2	act 1	act 2	SRMSE	MASE	SMAPE	SMAE
mean	✗	1,7	50	200	sigmoid	sigmoid	0.6332	0.7670	0.2516	4.4341
mean	✗	1	50	100	sigmoid	sigmoid	0.6334	0.7706	0.2545	4.2169
mean	✗	1	50	200	sigmoid	sigmoid	0.6342	0.7708	0.2548	4.2914
mean	✗	1	50	50	sigmoid	sigmoid	0.6344	0.7719	0.2545	4.4327
mean	✗	1	100	100	sigmoid	sigmoid	0.6351	0.7708	0.2513	4.2733
mean	✗	1	200	50	sigmoid	sigmoid	0.6358	0.7733	0.2523	4.3884
mean	✗	1	100	200	sigmoid	sigmoid	0.6359	0.7701	0.2525	4.3331
mean	✗	1,7	50	100	sigmoid	sigmoid	0.6359	0.7692	0.2532	4.5875
mean	✗	1	100	50	tanh	sigmoid	0.6371	0.7714	0.2554	4.3009
mean	✗	1,7	50	50	sigmoid	tanh	0.6372	0.7721	0.2535	4.5617
mean	✗	1	100	50	sigmoid	tanh	0.6374	0.7760	0.2554	4.3965
mean	✗	1	200	50	sigmoid	tanh	0.6375	0.7723	0.2539	4.4721
mean	✗	1	100	50	sigmoid	sigmoid	0.6375	0.7742	0.2514	4.2513
mean	✗	1,7	50	100	tanh	sigmoid	0.6377	0.7767	0.2630	4.3467
mean	✗	1	50	100	tanh	sigmoid	0.6380	0.7772	0.2563	4.3074
mean	✗	1	100	100	tanh	sigmoid	0.6383	0.7778	0.2566	4.3902
mean	✗	1	50	100	sigmoid	tanh	0.6384	0.7789	0.2588	4.2907
mean	✗	1	200	200	sigmoid	sigmoid	0.6386	0.7717	0.2514	4.4358
mean	✗	1,7	50	50	tanh	sigmoid	0.6388	0.7781	0.2620	4.5034
mean	✗	1,7	50	50	sigmoid	sigmoid	0.6389	0.7769	0.2559	4.5189
mean	✗	1	100	100	sigmoid	tanh	0.6391	0.7791	0.2552	4.2870
mean	✗	1,7	100	100	sigmoid	sigmoid	0.6393	0.7698	0.2544	4.6631
mean	✗	1	50	50	sigmoid	tanh	0.6393	0.7816	0.2587	4.2633
mean	✗	1,7	100	200	tanh	sigmoid	0.6394	0.7755	0.2600	4.2514
mean	✗	1	50	100	tanh	tanh	0.6394	0.7793	0.2554	4.3517
mean	✗	1	50	200	tanh	tanh	0.6397	0.7807	0.2571	4.3981
mean	✗	1	200	100	sigmoid	sigmoid	0.6401	0.7735	0.2512	4.3004
mean	✗	1,7	100	50	sigmoid	tanh	0.6404	0.7794	0.2588	4.5337
mean	✗	1	50	50	tanh	sigmoid	0.6406	0.7812	0.2590	4.3318
mean	✗	1	100	200	tanh	sigmoid	0.6406	0.7796	0.2583	4.5587
mean	✗	1,7	100	200	sigmoid	sigmoid	0.6409	0.7733	0.2549	4.6409
mean	✗	1,7	50	200	sigmoid	tanh	0.6410	0.7777	0.2590	4.5368
mean	✗	1,7	50	100	sigmoid	tanh	0.6410	0.7781	0.2623	4.5006
mean	✗	1,7	100	50	sigmoid	sigmoid	0.6411	0.7744	0.2571	4.4001
mean	✗	1,7	200	100	sigmoid	sigmoid	0.6411	0.7764	0.2520	4.3376
mean	✗	1	100	200	tanh	tanh	0.6415	0.7843	0.2641	4.3208
mean	✗	1	200	100	sigmoid	tanh	0.6416	0.7817	0.2547	4.3338
mean	✗	1,7	50	200	tanh	sigmoid	0.6420	0.7833	0.2607	4.3400
mean	✗	1,7	200	200	sigmoid	sigmoid	0.6421	0.7719	0.2515	4.5642
mean	✗	1	50	200	sigmoid	tanh	0.6421	0.7879	0.2616	4.2482
mean	✗	1,7	100	100	sigmoid	tanh	0.6421	0.7831	0.2585	4.4045
mean	✗	1	200	200	sigmoid	tanh	0.6422	0.7844	0.2575	4.4159
mean	✗	1,7	200	50	sigmoid	sigmoid	0.6423	0.7767	0.2529	4.6468
mean	✗	1	100	100	tanh	tanh	0.6434	0.7864	0.2583	4.2447
mean	✗	1	200	50	tanh	sigmoid	0.6434	0.7811	0.2625	4.2329

Continued on the next page

Table 12.8 – continued from the previous page

prep	W	w	hid 1	hid 2	act 1	act 2	SRMSE	MASE	SMAPE	SMAE
mean	✓	1	100	100	sigmoid	sigmoid	0.6436	0.7864	0.2565	4.2535
mean	✗	1	50	200	tanh	sigmoid	0.6439	0.7877	0.2606	4.3872
mean	✓	1	50	200	sigmoid	sigmoid	0.6441	0.7847	0.2592	4.2280
mean	✗	1	100	200	sigmoid	tanh	0.6443	0.7838	0.2583	4.3276
mean	✓	1	50	100	sigmoid	sigmoid	0.6446	0.7832	0.2599	4.3035
mean	✓	1	200	50	sigmoid	sigmoid	0.6447	0.7847	0.2563	4.2042
mean	✗	1	50	50	tanh	tanh	0.6447	0.7877	0.2682	4.3433
mean	✗	1	200	100	tanh	tanh	0.6450	0.7887	0.2627	4.2633
mean	✓	1,7	50	200	sigmoid	sigmoid	0.6451	0.7883	0.2625	4.0930
mean	✗	1	100	50	tanh	tanh	0.6451	0.7853	0.2603	4.4567
mean	✗	1,7	100	100	tanh	sigmoid	0.6455	0.7856	0.2606	4.5134
mean	✗	1	200	200	tanh	sigmoid	0.6456	0.7837	0.2597	4.4423
mean	✗	1,7	50	50	tanh	tanh	0.6458	0.7876	0.2599	4.5178
mean	✓	1,7	50	100	sigmoid	tanh	0.6460	0.7859	0.2632	4.1672
mean	✗	1,7	100	200	sigmoid	tanh	0.6461	0.7825	0.2634	4.4911
mean	✗	1,7	50	100	tanh	tanh	0.6462	0.7880	0.2641	4.6459
mean	✗	1,7	50	200	tanh	tanh	0.6462	0.7869	0.2609	4.6648
mean	✗	1,7	100	200	tanh	tanh	0.6462	0.7863	0.2675	4.4949
mean	✗	1	200	100	tanh	sigmoid	0.6464	0.7912	0.2705	4.3177
mean	✗	1,7	200	50	sigmoid	tanh	0.6467	0.7833	0.2574	4.4675
mean	✗	1	200	50	tanh	tanh	0.6470	0.7910	0.2673	4.4939
mean	✗	1,7	100	100	tanh	tanh	0.6470	0.7877	0.2656	4.5321
mean	✓	1	50	50	sigmoid	sigmoid	0.6471	0.7908	0.2604	4.2084
mean	✓	1,7	50	50	sigmoid	tanh	0.6472	0.7870	0.2606	4.1471
mean	✗	1,7	100	50	tanh	tanh	0.6472	0.7952	0.2692	4.4575
mean	✓	1	50	100	sigmoid	tanh	0.6474	0.7851	0.2584	4.3199
mean	✓	1	50	100	tanh	sigmoid	0.6474	0.7898	0.2689	4.1846
mean	✓	1,7	50	100	tanh	sigmoid	0.6475	0.7881	0.2685	4.3020
mean	✓	1	50	50	sigmoid	tanh	0.6478	0.7881	0.2544	4.2607
mean	✗	1,7	100	50	tanh	sigmoid	0.6481	0.7902	0.2610	4.3780
mean	✓	1	50	50	tanh	sigmoid	0.6486	0.7924	0.2674	4.2679
mean	✓	1	200	100	sigmoid	tanh	0.6492	0.7875	0.2634	4.2774
mean	✓	1	100	50	sigmoid	sigmoid	0.6494	0.7886	0.2645	4.2484
mean	✓	1	100	200	sigmoid	sigmoid	0.6498	0.7906	0.2589	4.2469
mean	✗	1,7	200	200	sigmoid	tanh	0.6499	0.7878	0.2622	4.3310
mean	✓	1,7	50	50	sigmoid	sigmoid	0.6500	0.7938	0.2657	4.3039
mean	✓	1	50	200	sigmoid	tanh	0.6504	0.7909	0.2602	4.2207
mean	✓	1	100	200	sigmoid	tanh	0.6506	0.7910	0.2613	4.2800
mean	✓	1,7	50	100	sigmoid	sigmoid	0.6511	0.7919	0.2657	4.4041
mean	✗	1	200	200	tanh	tanh	0.6511	0.7977	0.2689	4.4732
mean	✓	1	50	100	tanh	tanh	0.6512	0.7960	0.2645	4.3377
mean	✓	1	50	200	tanh	sigmoid	0.6516	0.7972	0.2685	4.0540
mean	✓	1,7	50	50	tanh	sigmoid	0.6519	0.7903	0.2668	4.4443
mean	✓	1,7	100	50	sigmoid	sigmoid	0.6521	0.7971	0.2681	4.1820
mean	✓	1,7	50	200	sigmoid	tanh	0.6523	0.7938	0.2626	4.2904
mean	✗	1,7	200	100	sigmoid	tanh	0.6525	0.7888	0.2681	4.6893
mean	✓	1	100	50	sigmoid	tanh	0.6533	0.7966	0.2595	4.0183
mean	✓	1,7	50	200	tanh	sigmoid	0.6533	0.7975	0.2681	4.5029
mean	✓	1	50	200	tanh	tanh	0.6542	0.7950	0.2631	4.0833
mean	✓	1,7	50	200	tanh	tanh	0.6543	0.7965	0.2649	4.3153
mean	✓	1,7	100	200	sigmoid	sigmoid	0.6545	0.7977	0.2669	4.4436
mean	✓	1	100	100	tanh	sigmoid	0.6549	0.8044	0.2705	4.0671
mean	✓	1	100	200	tanh	sigmoid	0.6551	0.8028	0.2761	4.2416
mean	✗	1,7	200	50	tanh	sigmoid	0.6551	0.7948	0.2636	4.6404

Continued on the next page

Table 12.8 – continued from the previous page

prep	W	w	hid 1	hid 2	act 1	act 2	SRMSE	MASE	SMAPE	SMAE
mean	✓	1,7	100	50	sigmoid	tanh	0.6560	0.7954	0.2636	4.2206
mean	✓	1	100	100	sigmoid	tanh	0.6562	0.7975	0.2618	4.1519
mean	✗	1,7	200	50	tanh	tanh	0.6567	0.7998	0.2699	4.5545
mean	✗	1,7	200	100	tanh	tanh	0.6569	0.8043	0.2733	4.3728
mean	✓	1	50	50	tanh	tanh	0.6570	0.8007	0.2653	4.2259
mean	✓	1	100	50	tanh	sigmoid	0.6570	0.8052	0.2724	4.5261
mean	✓	1,7	50	100	tanh	tanh	0.6571	0.8031	0.2652	4.4914
mean	✗	1,7	200	100	tanh	sigmoid	0.6573	0.7980	0.2681	4.6422
none	✗	1,7	100	200	sigmoid	sigmoid	0.6573	0.7848	0.2484	5.1542
none	✗	1,7	50	100	sigmoid	sigmoid	0.6574	0.7934	0.2517	5.1388
none	✗	1,7	50	50	sigmoid	tanh	0.6577	0.8004	0.2552	4.7704
none	✗	1,7	50	200	sigmoid	tanh	0.6589	0.7986	0.2539	4.6493
mean	✓	1,7	200	50	sigmoid	tanh	0.6589	0.8082	0.2723	4.3378
mean	✓	1	100	200	tanh	tanh	0.6595	0.8121	0.2761	4.1224
mean	✓	1,7	100	100	sigmoid	tanh	0.6595	0.8041	0.2686	4.3347
none	✓	1,7	50	50	sigmoid	tanh	0.6596	0.8034	0.2564	4.8001
mean	✓	1,7	100	200	tanh	sigmoid	0.6597	0.8112	0.2751	4.4776
mean	✓	1,7	50	50	tanh	tanh	0.6602	0.8096	0.2683	4.4564
mean	✓	1	200	50	sigmoid	tanh	0.6603	0.8045	0.2635	4.0448
mean	✓	1	200	200	sigmoid	sigmoid	0.6604	0.8021	0.2636	4.3724
none	✗	1,7	100	50	sigmoid	tanh	0.6606	0.7998	0.2615	4.7032
none	✗	1,7	50	200	tanh	sigmoid	0.6608	0.7928	0.2515	5.1196
mean	✓	1,7	100	100	sigmoid	sigmoid	0.6608	0.8071	0.2629	4.5108
none	✓	1,7	50	50	tanh	tanh	0.6612	0.8037	0.2581	4.5688
none	✗	1,7	50	100	sigmoid	tanh	0.6614	0.8007	0.2553	4.6345
mean	✓	1	100	50	tanh	tanh	0.6614	0.8157	0.2822	4.2699
none	✗	1,7	50	100	tanh	sigmoid	0.6614	0.7966	0.2514	5.0823
mean	✓	1	100	100	tanh	tanh	0.6615	0.8118	0.2712	4.1436
mean	✓	1,7	100	50	tanh	sigmoid	0.6617	0.8058	0.2779	4.3613
none	✗	1,7	50	50	sigmoid	sigmoid	0.6617	0.7999	0.2525	5.0012
mean	✓	1,7	200	100	sigmoid	tanh	0.6618	0.8035	0.2719	4.3350
none	✗	1,7	100	100	sigmoid	sigmoid	0.6619	0.7942	0.2526	5.0345
none	✓	1,7	50	100	sigmoid	tanh	0.6620	0.8009	0.2550	4.8093
none	✓	1,7	50	200	tanh	sigmoid	0.6621	0.8000	0.2560	4.7283
mean	✓	1,7	200	200	sigmoid	tanh	0.6623	0.8086	0.2757	4.2748
none	✗	1,7	50	200	sigmoid	sigmoid	0.6625	0.7945	0.2532	5.0650
mean	✓	1,7	100	100	tanh	sigmoid	0.6626	0.8122	0.2769	4.4570
mean	✓	1	200	100	tanh	sigmoid	0.6626	0.8168	0.2782	4.1931
mean	✗	1,7	200	200	tanh	sigmoid	0.6627	0.8014	0.2641	4.5469
none	✗	1,7	50	50	tanh	tanh	0.6630	0.8048	0.2556	4.8104
mean	✓	1	200	200	sigmoid	tanh	0.6632	0.8096	0.2677	4.1586
none	✗	1,7	200	50	sigmoid	tanh	0.6632	0.8008	0.2603	4.9081
none	✗	1,7	100	50	sigmoid	sigmoid	0.6636	0.7986	0.2545	4.8608
none	✓	1,7	100	50	sigmoid	sigmoid	0.6639	0.8076	0.2579	4.7159
none	✗	1,7	50	100	tanh	tanh	0.6639	0.8044	0.2577	5.0466
mean	✓	1,7	100	200	sigmoid	tanh	0.6639	0.8132	0.2712	4.4227
none	✓	1,7	50	200	tanh	tanh	0.6642	0.8027	0.2609	4.4203
mean	✓	1	200	100	sigmoid	sigmoid	0.6642	0.8085	0.2668	4.4720
none	✗	1,7	100	100	tanh	sigmoid	0.6644	0.7965	0.2531	5.0068
none	✗	1,7	50	200	tanh	tanh	0.6645	0.7999	0.2538	4.6420
none	✓	1,7	50	100	tanh	sigmoid	0.6645	0.8023	0.2546	4.3851
none	✗	1,7	200	50	sigmoid	sigmoid	0.6648	0.7988	0.2567	4.9076
none	✗	1	100	50	sigmoid	tanh	0.6648	0.8122	0.2606	4.9581
none	✗	1,7	200	100	sigmoid	sigmoid	0.6649	0.7989	0.2560	4.8138

Continued on the next page

Table 12.8 – continued from the previous page

prep	W	w	hid 1	hid 2	act 1	act 2	SRMSE	MASE	SMAPE	SMAE
none	✗	1	50	50	sigmoid	sigmoid	0.6649	0.8101	0.2572	4.8404
none	✓	1,7	50	50	tanh	sigmoid	0.6651	0.8051	0.2577	4.5621
none	✓	1	50	200	sigmoid	tanh	0.6656	0.8079	0.2591	4.1952
none	✗	1,7	100	200	tanh	sigmoid	0.6657	0.7978	0.2535	4.8453
mean	✓	1,7	100	50	tanh	tanh	0.6659	0.8149	0.2704	4.6022
none	✓	1,7	50	100	sigmoid	sigmoid	0.6661	0.8041	0.2531	4.9477
none	✓	1,7	50	200	sigmoid	tanh	0.6663	0.8120	0.2594	4.6685
none	✓	1,7	100	100	tanh	sigmoid	0.6666	0.8091	0.2628	5.2768
mean	✓	1,7	200	100	sigmoid	sigmoid	0.6670	0.8099	0.2760	4.2940
none	✗	1,7	100	100	tanh	tanh	0.6671	0.8108	0.2597	4.8309
none	✗	1,7	100	50	tanh	sigmoid	0.6672	0.7997	0.2539	5.0286
none	✗	1,7	100	100	sigmoid	tanh	0.6674	0.8050	0.2580	4.8289
none	✗	1	100	100	sigmoid	sigmoid	0.6676	0.8076	0.2568	4.9727
mean	✗	1,7	200	200	tanh	tanh	0.6677	0.8141	0.2717	4.4713
none	✓	1,7	100	200	tanh	sigmoid	0.6678	0.8126	0.2643	4.6446
mean	✓	1,7	200	200	sigmoid	sigmoid	0.6679	0.8147	0.2671	4.3496
mean	✓	1,7	200	50	sigmoid	sigmoid	0.6679	0.8194	0.2790	4.3870
none	✗	1	50	100	sigmoid	sigmoid	0.6681	0.8110	0.2572	4.9316
none	✓	1	50	50	sigmoid	tanh	0.6684	0.8124	0.2595	4.3342
none	✗	1,7	50	50	tanh	sigmoid	0.6685	0.8065	0.2547	5.0052
none	✓	1,7	50	50	sigmoid	sigmoid	0.6687	0.8100	0.2546	4.6597
none	✗	1	50	100	sigmoid	tanh	0.6687	0.8132	0.2615	4.6954
none	✗	1	100	200	sigmoid	tanh	0.6688	0.8135	0.2639	4.7043
none	✗	1	100	100	sigmoid	tanh	0.6690	0.8205	0.2635	4.5410
mean	✓	1,7	100	100	tanh	tanh	0.6692	0.8219	0.2818	4.0562
none	✗	1	100	50	tanh	sigmoid	0.6693	0.8112	0.2589	4.7877
none	✗	1,7	100	50	tanh	tanh	0.6699	0.8068	0.2579	4.8948
none	✗	1	50	200	sigmoid	sigmoid	0.6699	0.8123	0.2556	4.7741
none	✗	1	50	50	tanh	sigmoid	0.6700	0.8193	0.2604	4.9009
mean	✓	1,7	100	200	tanh	tanh	0.6700	0.8245	0.2830	4.4427
none	✓	1	50	200	tanh	sigmoid	0.6701	0.8061	0.2567	4.2916
none	✗	1	50	50	sigmoid	tanh	0.6701	0.8188	0.2626	4.5288
none	✗	1	50	100	tanh	sigmoid	0.6701	0.8139	0.2581	4.8769
none	✓	1	100	200	tanh	sigmoid	0.6703	0.8158	0.2620	4.5989
none	✗	1,7	100	200	sigmoid	tanh	0.6705	0.8115	0.2639	5.1302
none	✓	1,7	100	100	sigmoid	tanh	0.6706	0.8140	0.2626	4.5419
none	✗	1	200	50	sigmoid	sigmoid	0.6707	0.8109	0.2610	4.9643
none	✓	1,7	200	100	sigmoid	sigmoid	0.6707	0.8101	0.2589	4.7197
none	✓	1,7	50	200	sigmoid	sigmoid	0.6710	0.8096	0.2547	4.5170
none	✓	1	50	100	sigmoid	sigmoid	0.6712	0.8087	0.2544	4.2826
none	✗	1,7	200	100	sigmoid	tanh	0.6712	0.8091	0.2598	4.9249
none	✗	1,7	200	200	tanh	sigmoid	0.6714	0.8067	0.2600	4.8011
none	✓	1,7	50	100	tanh	tanh	0.6714	0.8142	0.2663	4.8587
none	✗	1,7	200	200	sigmoid	tanh	0.6715	0.8111	0.2603	4.7211
none	✗	1	100	50	sigmoid	sigmoid	0.6716	0.8173	0.2622	4.9709
none	✗	1	200	200	sigmoid	sigmoid	0.6716	0.8085	0.2557	4.7964
none	✓	1	50	100	sigmoid	tanh	0.6718	0.8140	0.2631	4.6755
none	✗	1,7	100	200	tanh	tanh	0.6720	0.8208	0.2671	4.6813
none	✓	1	100	50	tanh	sigmoid	0.6722	0.8166	0.2633	4.6014
none	✓	1	50	200	sigmoid	sigmoid	0.6722	0.8135	0.2586	4.2384
none	✗	1,7	200	200	sigmoid	sigmoid	0.6722	0.8019	0.2533	4.7803
none	✓	1	50	50	sigmoid	sigmoid	0.6722	0.8114	0.2570	4.5434
none	✓	1,7	100	50	sigmoid	tanh	0.6725	0.8094	0.2598	4.8181
none	✓	1	50	100	tanh	sigmoid	0.6726	0.8088	0.2572	4.7344

Continued on the next page

Table 12.8 – continued from the previous page

prep	W	w	hid 1	hid 2	act 1	act 2	SRMSE	MASE	SMAPE	SMAE
none	✗	1	50	200	sigmoid	tanh	0.6729	0.8186	0.2619	4.8868
none	✓	1	50	100	tanh	tanh	0.6730	0.8175	0.2651	4.3701
none	✓	1	50	50	tanh	sigmoid	0.6733	0.8195	0.2606	4.6979
none	✗	1	200	100	sigmoid	tanh	0.6733	0.8205	0.2643	4.9660
none	✗	1	200	50	tanh	tanh	0.6734	0.8215	0.2654	4.9993
mean	✓	1	200	200	tanh	sigmoid	0.6735	0.8245	0.2846	4.2759
none	✗	1	200	100	sigmoid	sigmoid	0.6736	0.8176	0.2622	4.8267
none	✓	1,7	100	50	tanh	sigmoid	0.6738	0.8100	0.2609	4.6431
none	✗	1	50	100	tanh	tanh	0.6741	0.8208	0.2640	4.6645
none	✓	1	50	200	tanh	tanh	0.6741	0.8213	0.2643	4.5226
none	✓	1,7	100	50	tanh	tanh	0.6741	0.8233	0.2718	4.7460
none	✓	1	100	50	tanh	tanh	0.6744	0.8236	0.2663	4.2808
none	✗	1	50	200	tanh	tanh	0.6745	0.8269	0.2655	4.8394
none	✗	1	200	50	sigmoid	tanh	0.6748	0.8301	0.2683	4.8639
none	✓	1,7	100	100	sigmoid	sigmoid	0.6749	0.8181	0.2579	4.6922
none	✓	1	50	50	tanh	tanh	0.6750	0.8191	0.2635	4.5569
none	✓	1	100	50	sigmoid	sigmoid	0.6751	0.8142	0.2619	4.5889
mean	✓	1	200	50	tanh	tanh	0.6752	0.8318	0.2846	4.5640
none	✗	1	50	200	tanh	sigmoid	0.6753	0.8209	0.2603	5.1089
none	✗	1	100	200	sigmoid	sigmoid	0.6754	0.8187	0.2593	4.8629
none	✓	1,7	200	50	sigmoid	tanh	0.6755	0.8189	0.2641	4.9483
none	✗	1	50	50	tanh	tanh	0.6755	0.8236	0.2640	4.8446
mean	✓	1	200	50	tanh	sigmoid	0.6758	0.8224	0.2752	4.4911
none	✗	1	100	50	tanh	tanh	0.6758	0.8288	0.2658	4.8771
none	✓	1	200	50	sigmoid	tanh	0.6761	0.8253	0.2670	4.5192
mean	✓	1	200	100	tanh	tanh	0.6761	0.8389	0.2880	4.1262
none	✓	1	100	50	sigmoid	tanh	0.6763	0.8290	0.2673	4.5684
none	✗	1	100	100	tanh	sigmoid	0.6763	0.8212	0.2607	4.8355
none	✓	1,7	100	200	tanh	tanh	0.6765	0.8270	0.2744	4.4728
mean	✓	1,7	200	100	tanh	sigmoid	0.6766	0.8313	0.2834	4.2363
mean	✓	1	200	200	tanh	tanh	0.6767	0.8395	0.2908	4.3291
none	✗	1	200	200	tanh	sigmoid	0.6767	0.8225	0.2624	5.0380
none	✗	1,7	200	100	tanh	sigmoid	0.6769	0.8131	0.2598	4.7858
none	✗	1	200	200	sigmoid	tanh	0.6772	0.8213	0.2657	4.8747
none	✓	1,7	200	50	sigmoid	sigmoid	0.6776	0.8193	0.2630	4.7197
none	✗	1	200	100	tanh	sigmoid	0.6778	0.8276	0.2642	4.9858
none	✓	1,7	100	100	tanh	tanh	0.6778	0.8239	0.2722	4.5582
none	✓	1	100	100	sigmoid	tanh	0.6778	0.8234	0.2665	4.3858
none	✓	1,7	200	200	sigmoid	tanh	0.6783	0.8265	0.2689	4.6317
none	✓	1,7	100	200	sigmoid	sigmoid	0.6783	0.8208	0.2621	4.7606
none	✗	1	100	200	tanh	sigmoid	0.6785	0.8246	0.2609	5.1252
none	✗	1,7	200	50	tanh	tanh	0.6786	0.8210	0.2649	4.9215
none	✓	1,7	100	200	sigmoid	tanh	0.6790	0.8162	0.2617	4.6958
mean	✓	1,7	200	50	tanh	sigmoid	0.6791	0.8387	0.2826	4.3090
mean	✓	1,7	200	100	tanh	tanh	0.6791	0.8410	0.2898	4.4289
none	✓	1	100	100	tanh	sigmoid	0.6801	0.8274	0.2639	4.9600
none	✓	1,7	200	200	sigmoid	sigmoid	0.6806	0.8174	0.2601	5.1222
none	✗	1,7	200	50	tanh	sigmoid	0.6808	0.8119	0.2598	5.0428
mean	✓	1,7	200	50	tanh	tanh	0.6811	0.8369	0.2830	4.2412
none	✗	1	100	100	tanh	tanh	0.6812	0.8372	0.2705	5.0316
none	✗	1,7	200	200	tanh	tanh	0.6813	0.8309	0.2763	4.3786
none	✓	1	100	200	sigmoid	sigmoid	0.6814	0.8203	0.2606	4.6243
none	✓	1	100	100	sigmoid	sigmoid	0.6815	0.8267	0.2622	4.2719
mean	✓	1,7	200	200	tanh	sigmoid	0.6821	0.8405	0.2878	4.2674

Continued on the next page

Table 12.8 – continued from the previous page

prep	W	w	hid 1	hid 2	act 1	act 2	SRMSE	MASE	SMAPE	SMAE
none	✓	1	200	100	tanh	sigmoid	0.6822	0.8332	0.2744	4.6371
none	✓	1	200	100	sigmoid	tanh	0.6823	0.8288	0.2703	4.5815
none	✗	1	200	50	tanh	sigmoid	0.6829	0.8325	0.2633	5.0731
none	✗	1,7	200	100	tanh	tanh	0.6832	0.8304	0.2678	5.1656
none	✗	1	100	200	tanh	tanh	0.6833	0.8359	0.2727	5.1120
none	✓	1	100	200	tanh	tanh	0.6835	0.8363	0.2799	4.3611
none	✓	1	200	200	sigmoid	tanh	0.6837	0.8319	0.2672	4.5338
none	✓	1	100	200	sigmoid	tanh	0.6839	0.8311	0.2700	4.4306
none	✓	1	100	100	tanh	tanh	0.6843	0.8344	0.2784	4.6438
none	✓	1,7	200	100	sigmoid	tanh	0.6844	0.8304	0.2758	4.5570
none	✓	1,7	200	200	tanh	sigmoid	0.6853	0.8345	0.2779	4.4728
none	✓	1,7	200	100	tanh	sigmoid	0.6859	0.8294	0.2733	4.8558
none	✗	1	200	100	tanh	tanh	0.6862	0.8415	0.2698	4.9113
none	✓	1	200	200	sigmoid	sigmoid	0.6864	0.8289	0.2631	4.3072
none	✓	1	200	50	sigmoid	sigmoid	0.6864	0.8328	0.2663	4.1429
none	✓	1	200	200	tanh	sigmoid	0.6867	0.8371	0.2714	4.7448
none	✓	1	200	100	sigmoid	sigmoid	0.6871	0.8310	0.2688	4.3116
none	✓	1,7	200	50	tanh	sigmoid	0.6906	0.8292	0.2667	4.4576
none	✗	1	200	200	tanh	tanh	0.6911	0.8459	0.2747	4.8603
none	✓	1	200	200	tanh	tanh	0.6927	0.8517	0.2864	4.8255
none	✓	1	200	100	tanh	tanh	0.6928	0.8451	0.2814	4.6350
none	✓	1	200	50	tanh	sigmoid	0.6933	0.8355	0.2686	4.2931
none	✓	1,7	200	100	tanh	tanh	0.6933	0.8474	0.2811	5.4386
mean	✓	1,7	200	200	tanh	tanh	0.6948	0.8647	0.2982	4.5443
none	✓	1,7	200	50	tanh	tanh	0.6972	0.8480	0.2792	4.6816
none	✓	1,7	200	200	tanh	tanh	0.6974	0.8568	0.2917	4.9079
none	✓	1	200	50	tanh	tanh	0.7026	0.8562	0.2821	4.8539
des	✗	1	50	100	sigmoid	tanh	0.7326	0.8984	0.2936	4.4774
des	✗	1,7	50	200	sigmoid	tanh	0.7338	0.9000	0.2956	4.6556
des	✗	1	100	50	sigmoid	sigmoid	0.7343	0.8970	0.2899	4.5121
des	✗	1	50	200	sigmoid	sigmoid	0.7345	0.8951	0.2861	4.5138
des	✗	1	50	50	sigmoid	tanh	0.7349	0.9028	0.2921	4.4989
des	✗	1,7	50	200	sigmoid	sigmoid	0.7360	0.9001	0.2973	4.6035
des	✗	1,7	50	50	sigmoid	sigmoid	0.7361	0.9022	0.2965	4.5307
des	✗	1	50	50	sigmoid	sigmoid	0.7366	0.9025	0.2907	4.4748
des	✗	1	100	100	sigmoid	sigmoid	0.7367	0.9038	0.2914	4.5028
des	✗	1	50	100	sigmoid	sigmoid	0.7369	0.8977	0.2906	4.5007
des	✗	1	50	200	sigmoid	tanh	0.7372	0.9055	0.2975	4.7472
des	✗	1,7	50	100	sigmoid	tanh	0.7374	0.9069	0.2945	4.4318
des	✗	1	100	200	sigmoid	tanh	0.7374	0.9033	0.2952	4.2983
des	✗	1	100	50	sigmoid	tanh	0.7374	0.9005	0.2936	4.4371
des	✗	1	100	200	sigmoid	sigmoid	0.7376	0.9026	0.2938	4.5899
des	✗	1	50	200	tanh	tanh	0.7378	0.9083	0.2943	4.6387
des	✗	1,7	50	50	tanh	tanh	0.7380	0.9079	0.3005	4.6975
des	✗	1,7	50	50	sigmoid	tanh	0.7382	0.9065	0.3017	4.5516
des	✗	1,7	50	200	tanh	sigmoid	0.7386	0.9100	0.2982	4.5472
des	✗	1,7	100	100	sigmoid	sigmoid	0.7387	0.9080	0.2963	4.6514
des	✗	1	100	100	sigmoid	tanh	0.7388	0.9085	0.2957	4.5183
des	✗	1	100	50	tanh	tanh	0.7389	0.9098	0.2966	4.3370
des	✗	1,7	50	100	sigmoid	sigmoid	0.7390	0.9024	0.2951	4.3818
des	✗	1,7	100	50	sigmoid	sigmoid	0.7391	0.9043	0.2976	4.7747
des	✗	1	200	200	sigmoid	sigmoid	0.7397	0.9089	0.2970	4.6290
des	✗	1	200	50	sigmoid	tanh	0.7404	0.9096	0.2951	4.5071
des	✗	1,7	100	200	sigmoid	tanh	0.7409	0.9128	0.3035	4.5849

Continued on the next page

Table 12.8 – continued from the previous page

prep	W	w	hid 1	hid 2	act 1	act 2	SRMSE	MASE	SMAPE	SMAE
des	✗	1	100	100	tanh	sigmoid	0.7412	0.9143	0.2937	4.6776
des	✗	1	200	50	sigmoid	sigmoid	0.7413	0.9110	0.2929	4.5654
des	✗	1	50	50	tanh	sigmoid	0.7420	0.9091	0.2914	4.5510
des	✗	1,7	50	100	tanh	sigmoid	0.7420	0.9119	0.2977	4.7511
des	✗	1,7	50	200	tanh	tanh	0.7422	0.9142	0.2998	4.9729
des	✗	1,7	200	50	sigmoid	tanh	0.7426	0.9127	0.3092	4.4379
des	✗	1	100	200	tanh	tanh	0.7428	0.9194	0.3027	4.4775
des	✗	1	50	100	tanh	tanh	0.7428	0.9129	0.2971	4.4503
des	✗	1	50	50	tanh	tanh	0.7431	0.9119	0.2981	4.7444
des	✗	1	200	100	sigmoid	sigmoid	0.7432	0.9143	0.2964	4.4946
des	✗	1,7	200	200	sigmoid	sigmoid	0.7433	0.9115	0.3012	4.5014
des	✗	1	100	100	tanh	tanh	0.7435	0.9169	0.3033	4.4084
des	✗	1,7	100	50	sigmoid	tanh	0.7436	0.9126	0.3000	4.5443
des	✗	1	200	100	sigmoid	tanh	0.7441	0.9162	0.2968	4.6982
des	✗	1,7	100	100	sigmoid	tanh	0.7444	0.9172	0.3011	4.4319
des	✗	1	200	200	sigmoid	tanh	0.7445	0.9178	0.3002	4.2875
des	✗	1,7	200	50	sigmoid	sigmoid	0.7445	0.9133	0.3053	4.7627
des	✗	1	200	50	tanh	tanh	0.7447	0.9171	0.3002	4.5348
des	✗	1	50	200	tanh	sigmoid	0.7448	0.9163	0.2948	4.3905
des	✗	1	100	200	tanh	sigmoid	0.7448	0.9161	0.2998	4.6139
des	✗	1,7	50	50	tanh	sigmoid	0.7448	0.9179	0.2996	4.7110
des	✗	1	200	100	tanh	tanh	0.7451	0.9193	0.3036	4.4406
des	✗	1	50	100	tanh	sigmoid	0.7462	0.9171	0.2953	4.5752
des	✗	1,7	100	200	tanh	sigmoid	0.7466	0.9179	0.3006	4.7750
des	✗	1,7	100	200	sigmoid	sigmoid	0.7466	0.9177	0.3000	4.6225
des	✗	1,7	50	100	tanh	tanh	0.7466	0.9204	0.3111	4.7409
des	✗	1,7	200	100	sigmoid	sigmoid	0.7470	0.9158	0.3025	4.5909
des	✗	1,7	100	200	tanh	tanh	0.7470	0.9231	0.3079	4.7602
des	✓	1	50	50	sigmoid	tanh	0.7471	0.9209	0.3116	4.5684
des	✗	1,7	200	200	sigmoid	tanh	0.7476	0.9196	0.3054	4.5565
des	✗	1,7	100	50	tanh	tanh	0.7480	0.9211	0.3062	4.6917
des	✗	1	200	100	tanh	sigmoid	0.7486	0.9253	0.2951	4.7033
des	✗	1,7	100	100	tanh	tanh	0.7487	0.9249	0.3147	5.0707
des	✗	1	100	50	tanh	sigmoid	0.7492	0.9172	0.2917	4.5789
des	✗	1,7	100	50	tanh	sigmoid	0.7494	0.9194	0.3023	4.7493
des	✓	1,7	50	50	sigmoid	tanh	0.7498	0.9222	0.3106	4.5340
des	✗	1,7	200	100	sigmoid	tanh	0.7500	0.9263	0.3082	4.5191
des	✗	1	200	200	tanh	tanh	0.7500	0.9234	0.3097	4.8229
des	✓	1,7	50	50	sigmoid	sigmoid	0.7508	0.9252	0.3104	4.6459
des	✓	1	50	200	sigmoid	tanh	0.7511	0.9372	0.3166	4.4460
des	✗	1	200	50	tanh	sigmoid	0.7511	0.9258	0.2964	4.3211
des	✓	1	50	100	sigmoid	sigmoid	0.7514	0.9266	0.3082	4.5031
des	✓	1	50	50	sigmoid	sigmoid	0.7522	0.9263	0.3059	4.5057
des	✓	1,7	50	200	sigmoid	tanh	0.7523	0.9299	0.3115	4.5026
des	✓	1	50	100	sigmoid	tanh	0.7538	0.9331	0.3140	4.3709
des	✓	1	50	200	sigmoid	sigmoid	0.7549	0.9303	0.3001	4.2435
des	✗	1	200	200	tanh	sigmoid	0.7557	0.9340	0.3009	4.7624
des	✓	1	50	50	tanh	sigmoid	0.7560	0.9305	0.3117	4.5868
des	✓	1	50	50	tanh	tanh	0.7562	0.9378	0.3148	4.5694
des	✓	1,7	50	100	sigmoid	tanh	0.7565	0.9306	0.3079	4.4842
des	✗	1,7	200	200	tanh	tanh	0.7574	0.9396	0.3262	4.7424
des	✓	1,7	100	200	sigmoid	tanh	0.7575	0.9329	0.3182	4.6615
des	✓	1,7	50	50	tanh	sigmoid	0.7575	0.9342	0.3149	4.8885
des	✓	1	50	200	tanh	sigmoid	0.7577	0.9386	0.3124	4.4181

Continued on the next page

Table 12.8 – continued from the previous page

prep	W	w	hid 1	hid 2	act 1	act 2	SRMSE	MASE	SMAPE	SMAE
des	✗	1,7	200	100	tanh	sigmoid	0.7577	0.9292	0.3095	4.7087
des	✓	1	50	100	tanh	sigmoid	0.7582	0.9369	0.3112	4.4732
des	✗	1,7	200	50	tanh	tanh	0.7584	0.9333	0.3102	4.9906
des	✓	1,7	50	200	tanh	tanh	0.7584	0.9407	0.3196	4.5704
des	✓	1	100	50	sigmoid	tanh	0.7584	0.9345	0.3213	4.5840
des	✓	1,7	50	100	tanh	tanh	0.7586	0.9353	0.3120	4.7226
des	✗	1,7	200	200	tanh	sigmoid	0.7590	0.9359	0.3090	4.4563
des	✓	1,7	50	200	sigmoid	sigmoid	0.7590	0.9352	0.3109	4.6855
des	✓	1	100	50	tanh	sigmoid	0.7607	0.9425	0.3163	4.3514
des	✗	1,7	100	100	tanh	sigmoid	0.7611	0.9396	0.3039	4.7126
des	✓	1,7	50	200	tanh	sigmoid	0.7613	0.9418	0.3133	4.7293
des	✓	1,7	100	50	sigmoid	tanh	0.7618	0.9402	0.3170	4.6383
des	✓	1	50	100	tanh	tanh	0.7619	0.9423	0.3213	4.6313
des	✓	1,7	100	100	sigmoid	tanh	0.7620	0.9429	0.3191	4.1822
des	✓	1	100	200	sigmoid	tanh	0.7621	0.9455	0.3115	4.4096
des	✓	1,7	100	50	sigmoid	sigmoid	0.7621	0.9352	0.3101	4.6160
des	✓	1,7	50	100	tanh	sigmoid	0.7622	0.9464	0.3178	4.5211
des	✓	1,7	50	50	tanh	tanh	0.7622	0.9438	0.3180	4.8608
des	✓	1	100	50	sigmoid	sigmoid	0.7624	0.9367	0.3082	4.5171
des	✓	1	50	200	tanh	tanh	0.7628	0.9475	0.3170	4.2732
des	✓	1	100	100	sigmoid	tanh	0.7633	0.9486	0.3143	4.4128
des	✓	1	200	200	sigmoid	tanh	0.7633	0.9481	0.3207	4.2824
des	✗	1,7	200	100	tanh	tanh	0.7638	0.9455	0.3171	4.4955
des	✓	1,7	50	100	sigmoid	sigmoid	0.7640	0.9375	0.3105	4.8011
des	✓	1,7	100	100	sigmoid	sigmoid	0.7640	0.9392	0.3089	4.6450
des	✓	1,7	100	50	tanh	sigmoid	0.7645	0.9447	0.3210	4.6117
des	✓	1,7	100	200	sigmoid	sigmoid	0.7653	0.9423	0.3158	4.4863
des	✓	1	200	100	sigmoid	sigmoid	0.7656	0.9474	0.3149	4.2482
des	✓	1,7	100	50	tanh	tanh	0.7658	0.9529	0.3225	4.7583
des	✓	1	200	50	sigmoid	sigmoid	0.7669	0.9484	0.3112	4.1700
des	✓	1,7	100	100	tanh	sigmoid	0.7676	0.9549	0.3209	4.4228
des	✓	1	100	100	sigmoid	sigmoid	0.7677	0.9449	0.3147	4.1730
des	✓	1,7	200	200	sigmoid	sigmoid	0.7679	0.9480	0.3229	4.9257
des	✓	1,7	200	100	sigmoid	tanh	0.7688	0.9500	0.3305	4.8873
des	✓	1,7	100	200	tanh	sigmoid	0.7696	0.9566	0.3260	4.4431
des	✓	1	200	50	sigmoid	tanh	0.7696	0.9516	0.3245	4.1714
des	✓	1	100	100	tanh	tanh	0.7705	0.9549	0.3259	4.3631
des	✓	1,7	100	100	tanh	tanh	0.7707	0.9540	0.3253	4.9396
des	✓	1,7	200	200	sigmoid	tanh	0.7708	0.9583	0.3241	5.0155
des	✓	1	100	200	tanh	tanh	0.7709	0.9567	0.3255	4.1881
des	✓	1	100	200	sigmoid	sigmoid	0.7711	0.9520	0.3113	4.2387
des	✓	1	100	100	tanh	sigmoid	0.7712	0.9591	0.3257	4.3101
des	✓	1	100	200	tanh	sigmoid	0.7723	0.9592	0.3198	4.0241
des	✓	1,7	200	50	sigmoid	sigmoid	0.7725	0.9519	0.3220	4.7144
des	✓	1,7	200	100	sigmoid	sigmoid	0.7727	0.9559	0.3212	4.4399
des	✓	1	200	200	sigmoid	sigmoid	0.7729	0.9518	0.3177	4.6238
des	✗	1,7	200	50	tanh	sigmoid	0.7729	0.9488	0.3060	4.7208
des	✓	1,7	200	50	sigmoid	tanh	0.7757	0.9566	0.3257	5.0847
des	✓	1,7	100	200	tanh	tanh	0.7768	0.9662	0.3285	4.7878
des	✓	1,7	200	100	tanh	sigmoid	0.7785	0.9670	0.3342	4.5324
des	✓	1	200	100	sigmoid	tanh	0.7798	0.9657	0.3235	4.5785
des	✓	1	200	100	tanh	tanh	0.7802	0.9697	0.3313	4.7621
des	✓	1,7	200	50	tanh	sigmoid	0.7805	0.9666	0.3334	4.7214
des	✓	1	100	50	tanh	tanh	0.7811	0.9714	0.3264	4.3774

Continued on the next page

Table 12.8 – continued from the previous page

prep	W	w	hid 1	hid 2	act 1	act 2	SRMSE	MASE	SMAPE	SMAE
des	✓	1	200	50	tanh	sigmoid	0.7811	0.9674	0.3231	4.4594
des	✓	1	200	100	tanh	sigmoid	0.7812	0.9723	0.3257	4.2226
des	✓	1	200	200	tanh	sigmoid	0.7813	0.9692	0.3207	4.2644
des	✓	1	200	50	tanh	tanh	0.7833	0.9711	0.3322	4.4318
des	✓	1,7	200	200	tanh	sigmoid	0.7852	0.9762	0.3298	4.5425
des	✓	1,7	200	50	tanh	tanh	0.7863	0.9795	0.3400	4.5396
des	✓	1,7	200	100	tanh	tanh	0.7863	0.9767	0.3274	5.0146
des	✓	1	200	200	tanh	tanh	0.7954	0.9911	0.3394	4.3520
des	✓	1,7	200	200	tanh	tanh	0.7964	0.9953	0.3401	4.9326

"none", "mean" and "des" indicate the type of preprocessing used (no preprocessing, mean adjustment, deseasonalisation.
W marks whether weather data was included for training purposes.
w indicate which of the previous days were used, 1 means the previous day, 1,7 means the previous day and the corresponding day from previous week.
"hid 1" and "hid 2" indicate the number of neurons in the first and the second hidden layer.
"act 1" and "act 2" indicate the activation function of neurons in the first and the second hidden layer.
Results are sorted in ascending order by SRMSE.

The experimental results of SLNs are summarised in Table 12.7. As we can see, mean adjustment is the most successful preprocessing method, followed by no preprocessing (only standardisation) and deseasonalisation as the worst. Also including the weather data is not beneficial. Larger window size helps the accuracy, together with smaller number of hidden neurons (50 or 100) and sigmoid activation function. The best and worst forecasts of the best performing SLN are summarised in Figure 12.5.

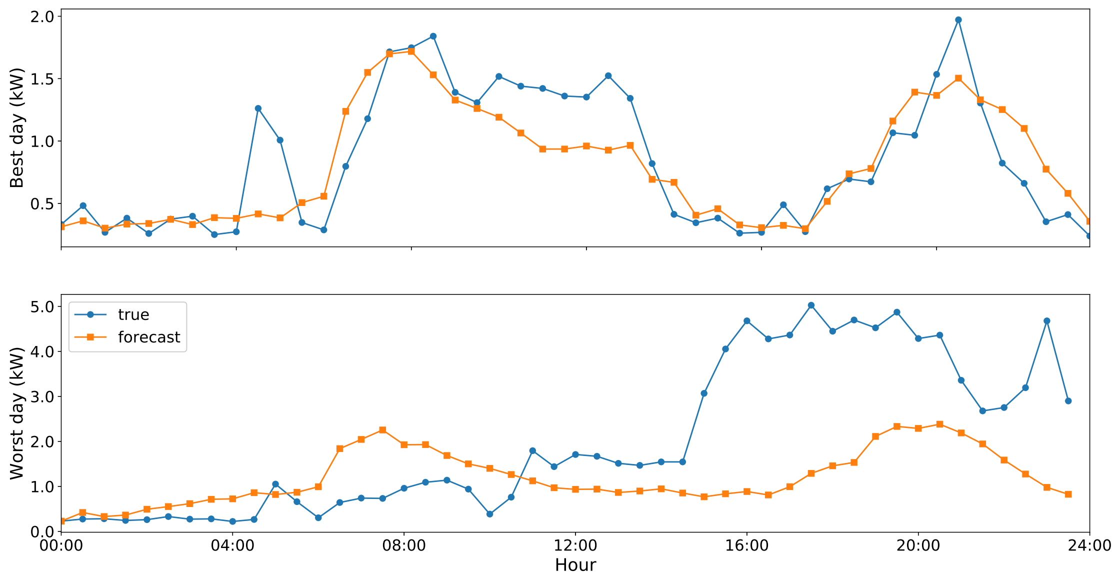

Figure 12.5: Best and worst load profile forecasts of the best SLN

Table 12.8 contains the experimental results of MLN with two hidden layers. Mean adjustment is the best preprocessing method, with no preprocessing (only standardisation) coming second and deseasonalisation as in the last place. Weather data again does not help produce more accurate forecasts, while larger window size yields only negligible improvements. It is best to use smaller number of neurons (50) in the first hidden layer followed by larger (100 or 200) number of neurons in the second hidden layer. Sigmoid activation function is again the better choice.

Figure 12.6 compares the best and worst forecasts of the best performing MLN with two hidden layers.

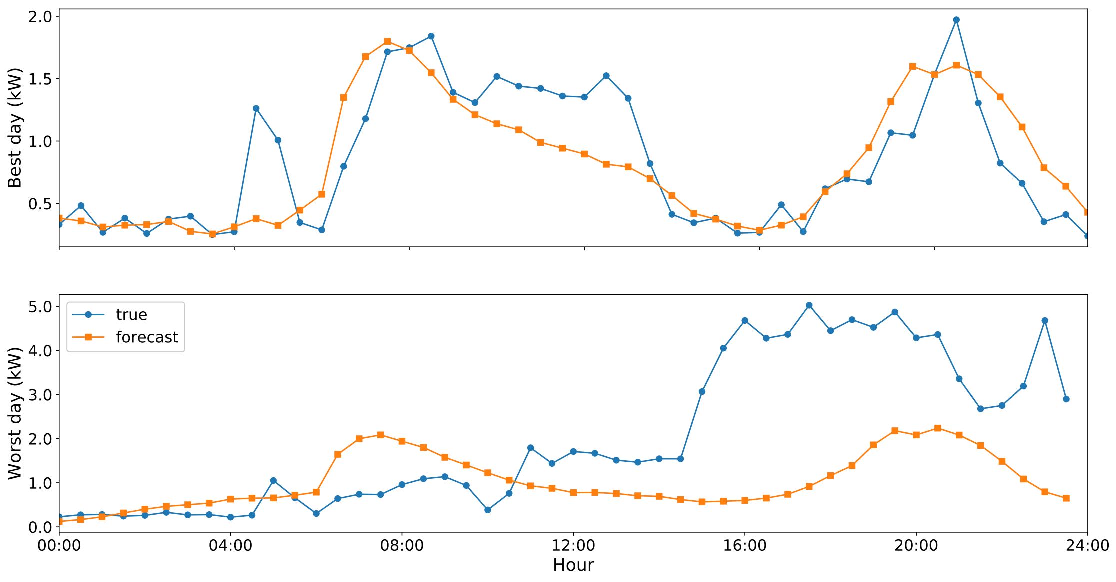

Figure 12.6: Best and worst load profile forecasts of the best MLN with one hidden layer

Overall neural networks seem less sensitive to hyperparameter tuning compared to ES, ARMA or ARMAX models. For example all networks are better than a naive forecasting method (MASE < 1). Also there are no MLN with unusually large SMAE.

12.5 Time complexity

In this section we summarize the approximate time complexity of the entirety of one monthly cross-validation process with respect to the type of model. The experiments were conducted using Intel Core i74712MQ processor with 8 GB of RAM. The listed times also include the time needed for all the preprocessing methods of a particular model. Entries in the following list are ordered from the fastest to the slowest:

(1): 10-20 minutes for SLNs

(2): 10-30 minutes for ES models

(3): 10-30 minutes for MLNs with 2 hidden layers

(4): 10-60 minutes for ARMA models

(5): 10-360 minutes (six hours) for ARMAX models

12.6 Ensemble learning

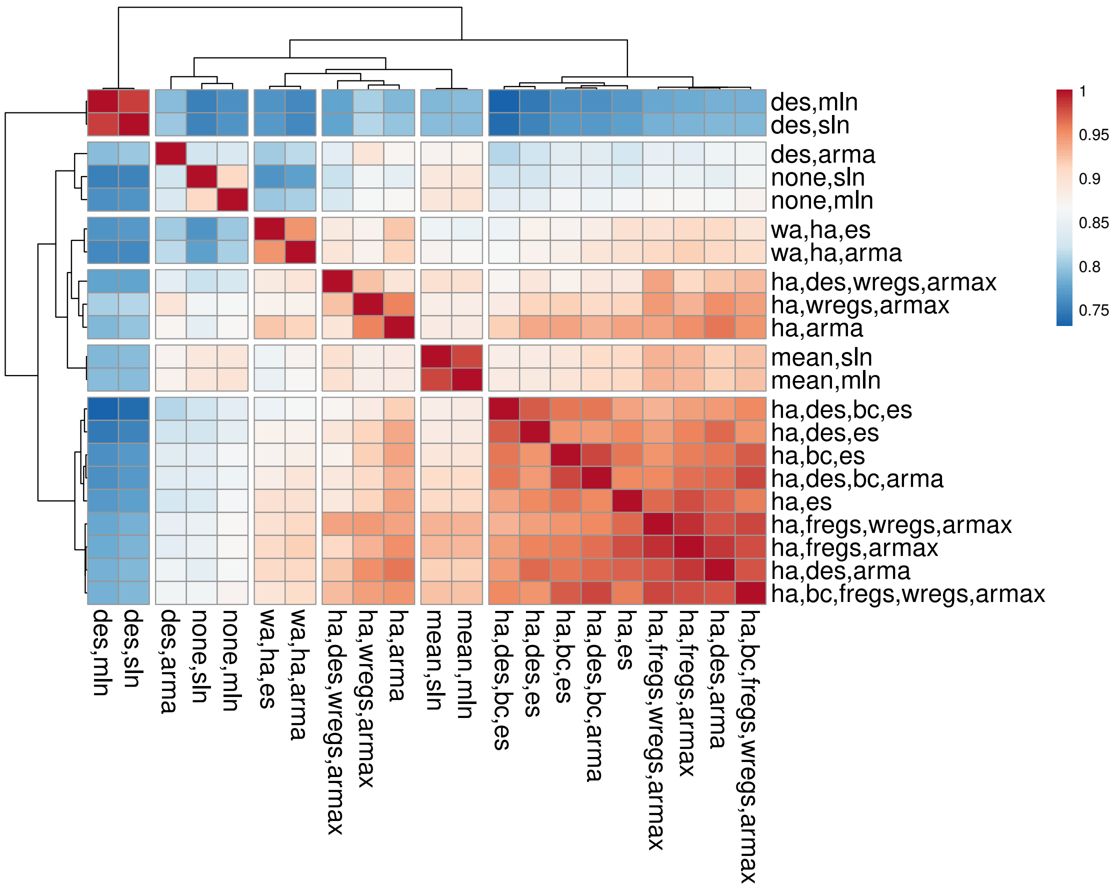

Figure 12.7: Clustering of models according to the absolute correlation of their residuals "es", "arma", "armax", "sln", "mln" indicate the type of model (ES, ARMA, ARMAX, SLN, MLN with two hidden layers).
"none", "mean", "des" describe the preprocessing methods for neural networks.
"wa", "ha", "des", "bc", "wregs", "fregs" describe the preprocessing methods for statistical models (ES, ARMA, ARMAX).
Ward clustering [93] with unsquared Euclidean distance was used to produce the clusters.
Produced using [94].

Ensemble learning is the practice of taking multiple algorithms and combining their output. If the individual outputs are combined appropriately, the performance of the ensemble is in many cases better than the performance of individual members [95].

There are two main approaches to ensemble learning: *static ensembles* and *dynamic ensembles* [96]. In static ensembles the output of the individual algorithms is combined in a way that does not include the input, e.g. in *ensemble averaging* the outputs of the ensemble members are combined in a linear fashion

Table 12.9: Experimental results for ensemble averaging

models	SRMSE	MASE	SMAPE	SMAE
ha,es des,arma ha,des,arma none, mln mean, mln	0.6157	0.7427	0.2382	4.3869
ha,des,es des,arma wa,ha,arma none, mln mean, mln	0.6157	0.7440	0.2388	4.4961
ha,es ha,des,es des,arma none, mln mean, mln	0.6158	0.7414	0.2375	4.4245
ha, des, es ha, arma mean, sln none, mln mean, mln	0.6159	0.7365	0.2345	4.4205
ha,arma ha,des,arma mean,sln none,mln mean,mln	0.6159	0.7380	0.2352	4.3828
ha,des,es wa,ha,arma ha,wregs,armax none,mln mean,mln	0.6160	0.7411	0.2369	4.4848
ha,arma ha,des,arma none,mln mean,mln	0.6162	0.7378	0.2352	4.4239

"es", "arma", "armax", "sln", "mln" indicate the type of model (ES, ARMA, ARMAX, SLN, MLN with two hidden layers).
"none", "mean" describe the preprocessing methods (only standardisation, mean adjustment) for neural networks (SLN, MLN)
"wa", "ha", "des", "wregs" describe the preprocessing methods (week adjustment, hour adjustment, deseasonalisation) for statistical models (ES, ARMA, ARMAX).
Results are sorted in ascending order by SRMSE.

using some type of weighting scheme. On the other hand, in dynamic ensembles the input is directly involved in the mechanism that combines the outputs of ensemble members. For example, *mixture of experts* combines the ensemble members nonlinearly via training an additional neural network on their outputs.

Table 12.10: Experimental results for the best combination of models

models	SRMSE	MASE	SMAPE	SMAE
none, mln mean, mln	0.6291	0.7530	0.2390	4.6818

”mln” means MLN with two hidden layers.
”none”, ”mean” describe the preprocessing methods for neural networks.

In this thesis we use ensemble averaging with weights equal to $1/k$, where k is the number of ensemble members. The motivation behind ensemble averaging is that when the ensemble members make different errors, the errors can be averaged out. We use the correlation of residuals of two models as the measure of the difference between their errors. As their correlation approaches 1, the two models make increasingly similar errors. The ideal ensemble would contain members with the lowest possible correlation with each other without sacrificing the accuracy of members. For this reason we chose the top five ES, ARMA, ARMAX models and the best SLN and MLN for each of the three preprocessing methods and constructed their correlation heatmap in Figure 12.7.

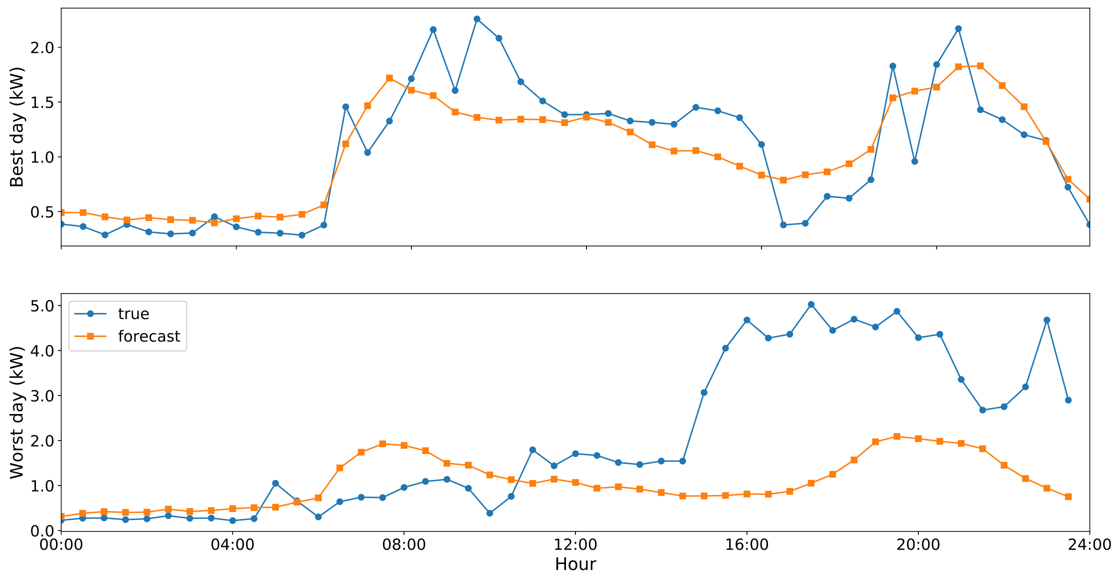

Figure 12.8: Best and worst load profile forecasts of the best ensemble

We chose the two most accurate ES, ARMA, ARMAX, SLN and MLN models, two most accurate models from each group in Figure 12.7 and also some less correlated models from groups in Figure 12.7 and tried every combination of up to five models, i.e. $k = 2, 3, 4, 5$. The top seven combinations are presented in Figure 12.9.

We can conclude that the combination of one MLN trained on mean-adjusted data and one MLN trained on standardised data is the most successful as it appears in all seven of the best combinations. The accuracy of an ensemble composed only of these two models is summarised in Table 12.10. All combinations also archived better performance than their respective members did individually. Figure 12.8 compares the best and worst forecasts of the best performing ensemble.

13. Conclusion

In this thesis we focused on forecasting daily energy load profiles of a family house. Our other objectives were describing the motivation and theoretical background behind various forecasting models, comparing the accuracy of various imputation methods and the influence of a number of preprocessing techniques on the accuracy of load profile forecasting.

We tested a variety of imputation methods by taking a portion of our data without any missing observations, artificially creating missing observations according to their distribution in the entire dataset and then imputing this modified portion of the data. The results indicate clear dominance of moving average smoothers over all other methods including various types of interpolation and ARMA models.

From among many preprocessing techniques under consideration the hour adjustment performs the best for ES, ARMA and also ARMAX models. For ARMA models also deseasonalisation improves the performance. However, for ARMAX models using Fourier regression, deseasonalisation is the worst preprocessing technique as it decreases their performance the most. Also for ARMAX models the best type of regression is Fourier regression beating weather regression and their combination. For SLN and MLN with two hidden layers the best type of preprocessing method is mean adjustment, followed by no preprocessing (only standardisation) and lastly deseasonalisation.

Regarding the comparison of preprocessing methods across different types of models, deseasonalisation is better than Fourier regression when modelling multiple seasonalities of energy consumption data, as evidenced by the performance of ARMA models using deseasonalisation versus the performance of ARMAX models using Fourier regression.

Weather data is not necessary for improving accuracy of models. However, the data does not include energy consumption for heating. On the other hand, south of France is relatively warm even throughout winter (compared to Prague), so the question of whether the weather data influence on energy consumption forecasting remains open.

When comparing the performance of the best model from each category, going from the most to the least accurate the order is ARMA trained on hour-adjusted, deseasonalised time series, mean-adjusted MLN, mean-adjusted SLN, ARMAX using Fourier regression trained on hour-adjusted data, hour-adjusted ES. However, there are multiple SLNs (3) and mainly MLNs (16) that are more accurate than both the second best ARMA model and the best ARMAX model. It is also important to note that SLNs and MLNs are less sensitive to preprocessing methods and parameter tuning than ES, ARMA and ARMAX models, as no neural network performed worse that a naive forecasting method. There were also no unpredictable flukes in terms of SMAE.

We also tried to improve the accuracy of the models by combining them into ensembles. This effort proved to be successful as many ensembles improved the accuracy measures of every individual member. The best combinations included mainly hour-adjusted ES models, deseasonalised ARMA models, hour-adjusted deseasonalised ARMA models, MLNs with only standardisation and mean-adjusted

MLNs.

13.1 Future work

A number of topics related to load profile forecasting that this thesis touched upon have been left out due to lack of time.

First of all, there are statistical and machine learning models that we did not experiment with. For example, a combination of ES and ARMA models designed to handle complex seasonalities [97] might be suitable for multiple seasonalities of energy consumption data. A different approach might be to use a long-short term memory network [98], a type of neural network that proved to be successful when forecasting time series.

One advantage of state-space-based statistical models, such as ES, ARMA and ARMAX, is that they are able to produce prediction intervals together with forecasts. Machine learning models are also capable of producing prediction intervals, although through methods different from state-space models. An optimisation algorithm designed to utilise load forecasts might benefit from those prediction intervals. It is also possible that, apart from the accuracy of the models, the width of prediction intervals might be an important criterion to consider.

Although we did examine the forecasting ability of ensembles of models, we did so in a static manner by combining the models into ensembles only after each of them had been estimated individually. A more dynamic approach would be to coordinate the respective estimation procedure of individual ensemble members based on their collective performance as en ensemble. For example, one can simultaneously train multiple neural networks in an ensemble by coordinating their training procedured based on the correlation of their errors [99]. The aim is to produce an ensemble of networks with negatively correlated errors to promote specialisation and cooperation. Another approach might be to train ensembles via genetic algorithms.

The scarcity of high quantity and quality energy consumption data has prevented us from comparing models across different datasets. When such data becomes available it would be interesting to investigate how do the results archived in this thesis translate to a different dataset.

In the meantime one can try to generate more data artificially and compare the forecasting abilities and adaptability to real-world conditions of models trained on artificially generated data with those trained on real-world data.

When high quantity data is not available, a model that is able to produce reasonable forecasts from a small dataset might be preferable, especially in practice, to a highly accurate model requiring a lot of observation. For example, in real-life scenarios we expect the model to have access to only a couple of days or weeks worth of observations. For this reason one might examine the evolution of performance and adaptability of models as they are presented with new data, starting from datasets with small number of observations and gradually increasing the size of datasets.

It is also important to note that real weather data that was used in this thesis may have introduced unwanted bias to forecasts. It can be argued that one should opt for using historical forecasts of weather data instead of real weather data. The comparison between forecasts using real weather data and forecasts using historical weather forecasts may shed some light on the significance of weather in energy consumption forecasting.

Bibliography

[1] Vehbi C Gungor, Dilan Sahin, Taskin Kocak, Salih Ergut, Concettina Buccella, Carlo Cecati, and Gerhard P Hancke. Smart grid technologies: Communication technologies and standards. *IEEE transactions on Industrial informatics*, 7(4):529–539, 2011.

[2] Hassan Farhangi. The path of the smart grid. *IEEE power and energy magazine*, 8(1), 2010.

[3] Antimo Barbato, Antonio Capone, Marta Rodolfi, and Davide Tagliaferri. Forecasting the usage of household appliances through power meter sensors for demand management in the smart grid. In *Smart Grid Communications (SmartGridComm), 2011 IEEE International Conference on*, pages 404–409. IEEE, 2011.

[4] James Douglas Hamilton. *Time series analysis*, volume 2. Princeton university press Princeton, 1994.

[5] Mariano Ventosa, Alvaro Baıllo, Andrés Ramos, and Michel Rivier. Electricity market modeling trends. *Energy policy*, 33(7):897–913, 2005.

[6] Luis Martín, Luis F Zarzalejo, Jesús Polo, Ana Navarro, Ruth Marchante, and Marco Cony. Prediction of global solar irradiance based on time series analysis: Application to solar thermal power plants energy production planning. *Solar Energy*, 84(10):1772–1781, 2010.

[7] Roy Billinton, Hua Chen, and R Ghajar. Time-series models for reliability evaluation of power systems including wind energy. *Microelectronics Reliability*, 36(9):1253–1261, 1996.

[8] Soma Shekara Sreenadh Reddy Depuru, Lingfeng Wang, and Vijay Devabhaktuni. Smart meters for power grid: Challenges, issues, advantages and status. *Renewable and sustainable energy reviews*, 15(6):2736–2742, 2011.

[9] Shichao Zhang, Chengqi Zhang, and Qiang Yang. Data preparation for data mining. *Applied Artificial Intelligence*, 17(5-6):375–381, 2003.

[10] James D Hamilton. State-space models. *Handbook of econometrics*, 4:3039–3080, 1994.

[11] Rob Hyndman, Anne B Koehler, J Keith Ord, and Ralph D Snyder. *Forecasting with exponential smoothing: the state space approach.* Springer Science & Business Media, 2008.

[12] Tao Hong. *Short term electric load forecasting.* North Carolina State University, 2010.

[13] Ryszard S Michalski, Jaime G Carbonell, and Tom M Mitchell. *Machine learning: An artificial intelligence approach.* Springer Science & Business Media, 2013.

[14] William G Baxt. Application of artificial neural networks to clinical medicine. *The lancet*, 346(8983):1135–1138, 1995.

[15] Robert R Trippi and Efraim Turban. *Neural networks in finance and investing: Using artificial intelligence to improve real world performance.* McGraw-Hill, Inc., 1992.

[16] Sushmito Ghosh and Douglas L Reilly. Credit card fraud detection with a neural-network. In *System Sciences, 1994. Proceedings of the Twenty-Seventh Hawaii International Conference on*, volume 3, pages 621–630. IEEE, 1994.

[17] Dean A Pomerleau. Efficient training of artificial neural networks for autonomous navigation. *Neural Computation*, 3(1):88–97, 1991.

[18] Henrik Madsen. *Time series analysis.* CRC Press, 2007.

[19] R Dennis Cook and Sanford Weisberg. *Residuals and influence in regression.* New York: Chapman and Hall, 1982.

[20] Anna Mikusheva and Paul Schrimpf. Stationarity, lag operator, arma, and covariance structure. Available at `https://ocw.mit.edu/courses/economics/14-384-time-series-analysis-fall-2013/lecture-notes/MIT14_384F13_lec1.pdf`, 2009.

[21] Rob J Hyndman. Moving averages. In *International Encyclopedia of Statistical Science*, pages 866–869. Springer, 2011.

[22] Robert B Cleveland, William S Cleveland, and Irma Terpenning. Stl: A seasonal-trend decomposition procedure based on loess. *Journal of Official Statistics*, 6(1):3, 1990.

[23] Robert Alan Yaffee and Monnie McGee. *An introduction to time series analysis and forecasting: with applications of SAS® and SPSS®.* Academic Press, 2000.

[24] James W Taylor. An evaluation of methods for very short-term load forecasting using minute-by-minute british data. *International Journal of Forecasting*, 24(4):645–658, 2008.

[25] Nazih Abu-Shikhah, Fawwaz Elkarmi, and Osama M Aloquili. Medium-term electric load forecasting using multivariable linear and non-linear regression. *Smart Grid and Renewable Energy*, 2(02):126, 2011.

[26] MS Kandil, Sm M El-Debeiky, and NE Hasanien. Long-term load forecasting for fast developing utility using a knowledge-based expert system. *IEEE transactions on Power Systems*, 17(2):491–496, 2002.

[27] Gerwin Hoogsteen, Albert Molderink, Johann L Hurink, and Gerard JM Smit. Generation of flexible domestic load profiles to evaluate demand side management approaches. In *Energy Conference (ENERGYCON), 2016 IEEE International*, pages 1–6. IEEE, 2016.

[28] Georges Hebrail and Alice Berard. Individual household electric power consumption data set. Available at `https://archive.ics.uci.edu/ml/datasets/individual+household+electric+power+consumption`, 2013.

[29] Georges Hébrail. Personal communication, 2016.

[30] Mahelet G Fikru and Luis Gautier. The impact of weather variation on energy consumption in residential houses. *Applied Energy*, 144:19–30, 2015.

[31] Weather underground api. Available at `https://www.wunderground.com/weather/api/`, 2017.

[32] Alex D Papalexopoulos and Timothy C Hesterberg. A regression-based approach to short-term system load forecasting. *IEEE Transactions on Power Systems*, 5(4):1535–1547, 1990.

[33] James W Taylor. Short-term electricity demand forecasting using double seasonal exponential smoothing. *Journal of the Operational Research Society*, 54(8):799–805, 2003.

[34] S Sp Pappas, L Ekonomou, P Karampelas, DC Karamousantas, SK Katsikas, GE Chatzarakis, and PD Skafidas. Electricity demand load forecasting of the hellenic power system using an arma model. *Electric Power Systems Research*, 80(3):256–264, 2010.

[35] D Datta, SA Tassou, and D Marriott. Application of neural networks for the prediction of the energy consumption in a supermarket. In *Proceedings of the International Conference CLIMA*, pages 98–107, 2000.

[36] Soteris A Kalogirou and Milorad Bojic. Artificial neural networks for the prediction of the energy consumption of a passive solar building. *Energy*, 25(5):479–491, 2000.

[37] J Douglas Balcomb. *Passive solar buildings*, volume 7. MIT press, 1992.

[38] SM Sulaiman, P Aruna Jeyanthy, and D Devaraj. Artificial neural network based day ahead load forecasting using smart meter data. In *Power and Energy Systems: Towards Sustainable Energy (PESTSE), 2016 Biennial International Conference on*, pages 1–6. IEEE, 2016.

[39] Alberto Hernandez Neto and Flávio Augusto Sanzovo Fiorelli. Comparison between detailed model simulation and artificial neural network for forecasting building energy consumption. *Energy and buildings*, 40(12):2169–2176, 2008.

[40] Drury B Crawley, Linda K Lawrie, Frederick C Winkelmann, Walter F Buhl, Y Joe Huang, Curtis O Pedersen, Richard K Strand, Richard J Liesen, Daniel E Fisher, Michael J Witte, et al. Energyplus: creating a new-generation building energy simulation program. *Energy and buildings*, 33(4):319–331, 2001.

[41] Chris Chatfield. *Time-series forecasting*. CRC Press, 2000.

[42] Gaëtan Libert, Liang Wang, and Bao Liu. An innovation state space approach for time series forecasting. *Journal of Time Series Analysis*, 14(6):589–601, 1993.

[43] James Durbin and Siem Jan Koopman. *Time series analysis by state space methods*, volume 38. OUP Oxford, 2012.

[44] Rudolph Emil Kalman et al. A new approach to linear filtering and prediction problems. *Journal of basic Engineering*, 82(1):35–45, 1960.

[45] Sergio Bittanti, Alan J Laub, and Jan C Willems. *The Riccati Equation.* Springer Science & Business Media, 2012.

[46] David A Freedman. *Statistical models: theory and practice.* cambridge university press, 2009.

[47] Dmitry Panchenko. Properties of mle: consistency, asymptotic normality. fisher information. Available at `https://ocw.mit.edu/courses/mathematics/18-443-statistics-for-applications-fall-2006/lecture-notes/lecture3.pdf`, 2006.

[48] Rob J Hyndman and George Athanasopoulos. *Forecasting: principles and practice.* OTexts, 2014.

[49] Robert G Brown. Exponential smoothing for predicting demand. In *Operations Research*, volume 5, pages 145–145. Inst Operations Research Management Sciences 901 Elkridge Landing Rd, ste 400, Linthicum Hts, MD 21090-2909, 1957.

[50] Charles C Holt. Forecasting seasonals and trends by exponentially weighted moving averages. *International journal of forecasting*, 20(1):5–10, 2004.

[51] Everette S Gardner Jr and ED McKenzie. Forecasting trends in time series. *Management Science*, 31(10):1237–1246, 1985.

[52] James W Taylor. Exponential smoothing with a damped multiplicative trend. *International journal of Forecasting*, 19(4):715–725, 2003.

[53] Peter R Winters. Forecasting sales by exponentially weighted moving averages. *Management science*, 6(3):324–342, 1960.

[54] Hirotugu Akaike. A new look at the statistical model identification. *IEEE transactions on automatic control*, 19(6):716–723, 1974.

[55] George EP Box, Gwilym M Jenkins, Gregory C Reinsel, and Greta M Ljung. *Time series analysis: forecasting and control.* John Wiley & Sons, 2015.

[56] George EP Box and David A Pierce. Distribution of residual autocorrelations in autoregressive-integrated moving average time series models. *Journal of the American statistical Association*, 65(332):1509–1526, 1970.

[57] Peter Whittle. *Hypothesis testing in time series analysis*, volume 4. Almqvist & Wiksells, 1951.

[58] Ruey Tsay. State space model and kalman filter. Available at http://faculty.chicagobooth.edu/ruey.tsay/teaching/uts/lec12-08.pdf, 2008.

[59] Denis Kwiatkowski, Peter CB Phillips, Peter Schmidt, and Yongcheol Shin. Testing the null hypothesis of stationarity against the alternative of a unit root: How sure are we that economic time series have a unit root? *Journal of econometrics*, 54(1-3):159–178, 1992.

[60] Denise R Osborn, Alice PL Chui, Jeremy P Smith, and Chris R Birchenhall. Seasonality and the order of integration for consumption. *Oxford Bulletin of Economics and Statistics*, 50(4):361–377, 1988.

[61] Stuart Russell and Peter Norvig. Artificial intelligence: A modern approach. *Artificial Intelligence. Prentice-Hall, Egnlewood Cliffs*, 25:27, 1995.

[62] George AF Seber and Alan J Lee. *Linear regression analysis*, volume 936. John Wiley & Sons, 2012.

[63] David Morin. Fourier analysis. Available at http://www.people.fas.harvard.edu/~djmorin/Fourier.pdf, 2009.

[64] Stephen Pollock. Maximum-likelihood estimation of the classical linear model. Available at http://www.le.ac.uk/users/dsgp1/COURSES/MATHSTAT/13mlreg.pdf.

[65] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[66] Warren S Sarle. Neural networks and statistical models. 1994.

[67] Michael A Nielsen. Neural networks and deep learning. Available at http://neuralnetworksanddeeplearning.com, 2015.

[68] Dale Purves, Roberto Cabeza, Scott A Huettel, Kevin S LaBar, Michael L Platt, Marty G Woldorff, and Elizabeth M Brannon. *Cognitive Neuroscience*. Sunderland: Sinauer Associates, Inc, 2008.

[69] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991.

[70] Stephen Judd. Learning in networks is hard. In *Proceedings of the First International Conference on Neural Networks*, pages 685–692. IEEE San Diego, CA, 1987.

[71] Sebastian Ruder. An overview of gradient descent optimization algorithms. Available at http://sebastianruder.com/optimizing-gradient-descent/index.html#gradientdescentvariants, 2016.

[72] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.

[73] Tom M Apostol. *Mathematical analysis*, volume 2. Addison-Wesley Reading, MA, 1974.

[74] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[75] Lutz Prechelt. Automatic early stopping using cross validation: quantifying the criteria. *Neural Networks*, 11(4):761–767, 1998.

[76] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Aistats*, volume 9, pages 249–256, 2010.

[77] Eric Zivot and Jiahui Wang. Rolling analysis of time series. In *Modeling Financial Time Series with S-Plus®*, pages 299–346. Springer, 2003.

[78] Rob J Hyndman and Anne B Koehler. Another look at measures of forecast accuracy. *International journal of forecasting*, 22(4):679–688, 2006.

[79] J Scott Armstrong. Combining forecasts. In *Principles of forecasting*, pages 417–439. Springer, 2001.

[80] Sylvain Arlot, Alain Celisse, et al. A survey of cross-validation procedures for model selection. *Statistics surveys*, 4:40–79, 2010.

[81] Sebastian Gottwalt, Wolfgang Ketter, Carsten Block, John Collins, and Christof Weinhardt. Demand side management—a simulation of household behavior under variable prices. *Energy policy*, 39(12):8163–8174, 2011.

[82] George EP Box and David R Cox. An analysis of transformations. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 211–252, 1964.

[83] Li Pengfei. Box-cox transformations: An overview. Available at `https://www.ime.usp.br/~abe/lista/pdfm9cJKUmFZp.pdf`, 2005.

[84] Norman R Draper and David R Cox. On distributions and their transformation to normality. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 472–476, 1969.

[85] Guoqiang Zhang, B Eddy Patuwo, and Michael Y Hu. Forecasting with artificial neural networks:: The state of the art. *International journal of forecasting*, 14(1):35–62, 1998.

[86] A Rogier T Donders, Geert JMG van der Heijden, Theo Stijnen, and Karel GM Moons. Review: a gentle introduction to imputation of missing values. *Journal of clinical epidemiology*, 59(10):1087–1091, 2006.

[87] Steffen Moritz. Time series missing value imputation. Available at `https://cran.r-project.org/web/packages/imputeTS/imputeTS.pdf`, 2017.

[88] Erik Meijering. A chronology of interpolation: From ancient astronomy to modern signal and image processing. *Proceedings of the IEEE*, 90(3):319–342, 2002.

[89] Carl De Boor, Carl De Boor, Etats-Unis Mathématicien, Carl De Boor, and Carl De Boor. *A practical guide to splines*, volume 27. Springer-Verlag New York, 1978.

[90] Russell W Stineman. A consistently well-behaved method of interpolation. *Creative Computing*, 6(7):54–57, 1980.

[91] Rob J Hyndman and Yeasmin Khandakar. Automatic time series for forecasting: the forecast package for R. Technical report, Monash University, Department of Econometrics and Business Statistics, 2007.

[92] François Chollet. Keras. Available at `https://github.com/fchollet/keras`, 2015.

[93] Joe H Ward Jr. Hierarchical grouping to optimize an objective function. *Journal of the American statistical association*, 58(301):236–244, 1963.

[94] Tauno Metsalu and Jaak Vilo. Clustvis: a web tool for visualizing clustering of multivariate data using principal component analysis and heatmap. *Nucleic acids research*, 43(W1):W566–W570, 2015.

[95] Gavin Brown. Ensemble learning. In *Encyclopedia of Machine Learning*, pages 312–320. Springer, 2011.

[96] Simon Haykin and Neural Network. A comprehensive foundation. *Neural Networks*, 2(2004):41, 2004.

[97] Alysha M De Livera, Rob J Hyndman, and Ralph D Snyder. Forecasting time series with complex seasonal patterns using exponential smoothing. *Journal of the American Statistical Association*, 106(496):1513–1527, 2011.

[98] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[99] Yong Liu and Xin Yao. Ensemble learning via negative correlation. *Neural networks*, 12(10):1399–1404, 1999.

List of Abbreviations

ACF	autocorrelation function
ADAM	adaptive moment estimation
API	application programming interface
AR	autoregressive
ARIMA	autoregressive integrated moving average
ARIMAX	autoregressive integrated moving average with exogenous inputs
ARMA	autoregressive moving average
ES	exponential smoothing
LOESS	local regression
MAE	mean absolute error
MAPE	mean absolute percentage error
MASE	mean absolute scaled error
MLN	multilayer (neural) network
MSE	mean squared error
RMSE	root mean squared error
SARIMA	seasonal autoregressive integrated moving average
SARIMAX	seasonal autoregressive integrated moving average with exogenous inputs
SMAE	scaled mean absolute error
SMAPE	scaled mean absolute percentage error
SRMSE	scaled root mean squared error
STL	seasonal-trend decomposition procedure based on LOESS

Attachments

The DVD attached to this thesis contains two folders and a file `info.txt` containing this information. The first folder `data` contains energy consumption data and weather data used for training and testing the models. The results of experiments are also located in this folder. The second folder `code` contains source code used for data preparation, imputation and experiments.

Data

All the data is split into several folders:

- `raw` contains raw energy consumption data
- `weather` contains raw weather data obtained via [31]
- `imp` contains the results of various imputation methods
- `train_test` contains preprocessed data split into training and testing sets ready to be used in experiments
- `es` contains the results for ES models and monthly cross-validation
- `es_week` contains the results for ES models and weekly cross-validation
- `arma` contains the results for ARMA models
- `armax` contains the results for ARMAX models
- `sln` contains the results for MLN with one hidden layer
- `mln` contains the results for MLN with two hidden layers
- `ensemble` contains the results for ensemble learning

Source code

Source code is divided into following files:

- `dataprep.py` contains functions used for data manipulation and preparation
- `imputation.py` contains functions for imputation
- `performance.py` contains functions for performance measures and performance evaluation
- `workspace.py` serves as a backbone of the whole program and utilizes `dataprep.py`, `imputation.py` and `performance.py` in order to download weather data, impute energy consumption data, prepare training and testing sets, evaluate the results of experiments and build tables and plots.
- `dataprep.R` contains functions used for loading and saving data for R scripts

- `es.R` sets up ES models and performs experiments
- `arma.R` sets up ARMA and ARMAX models and performs experiments
- `mln.py` sets up MLN models and performs experiments