Master's thesis

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

Daniel Aunan Bolstad

Interpretation of Electrical Load Forecasts using Explainable Artificial Intelligence

A day-ahead load forecasting case study of the NO1 price region

Master's thesis in Energy and the Environment
Supervisor: Ümit Cali
June 2021

NTNU
Norwegian University of Science and Technology

Daniel Aunan Bolstad

Interpretation of Electrical Load Forecasts using Explainable Artificial Intelligence

A day-ahead load forecasting case study of the NO1 price region

Master's thesis in Energy and the Environment
Supervisor: Ümit Cali
June 2021

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Electric Power Engineering

NTNU
Norwegian University of
Science and Technology

Preface

The following master thesis is the second part for the degree of master of science in electric power engineering. The work is the continuation of the project thesis carried out during the previous semester. The following thesis assumes that the reader is somewhat familiar with the basic concepts of machine learning, which were introduced in the project thesis (Bolstad, 2020).

Thanks to my supervisor, Ümit Cali for being available whenever I needed guidance, even outside business hours.

Daniel Aunan Bolstad, Trondheim 2021

Abstract

Electrical load forecasts are used by a wide number of power system participants in multiple time horizons, ranging from minutes ahead to several years ahead. Forecasts by machine learning models offer very high accuracy, but are so-called black boxes and do not give any reasoning for their decisions. Explainable artificial intelligence methods provide a means for peeking inside the black box to understand the model better.

The goal of the thesis was to investigate how explainable artificial intelligence can be used to interpret and improve electrical load forecasts made by a machine learning model. A convolutional neural network for day-ahead load forecasting was developed for the Norwegian price zone, NO1. The framework of Shapley additive explanations was used during model development for feature selection and debugging purposes. This framework was also used for explanations of selected forecasts made in testing of the final model. Local explanations were made by, essentially, comparing the forecast in question to other similar days through background data selection. During model development it was found that the explanations led to increased model performance and understanding. Moreover, the explanations made for the forecasts during testing proved to be intuitive and gave insight of the underlying causes of the forecasts.

The findings of the thesis therefore suggest that explainable artificial intelligence methods are well suited for electrical load forecasting models. However, the intuitiveness of the explanations rely on using similar forecasts for comparison. More research is needed on the implications of the choice of background data to make any conclusive statements.

Sammendrag

Forbruksprognoser i elektriske kraftsystemer er viktige for flere aktører over mange forskjellige tidshorisonter. Prognoser laget med anvendelse av maskinlære eller dyp lære, har ved flere anledninger vist seg å være mer nøyaktige enn konvensjonelle metoder, men er basert på "black box"-modeller som ikke er praktisk mulig å forstå. Forklarbar kunstig intelligens har som hensikt å gi innsyn i den underliggende modellen. Dette gir mulighet for videre utvikling av modellen og økt tiltro til den produserte prognosen.

Oppgaven utforsker bruken av forklarbar kunstig intelligens for å forstå og forbedre kortsiktige forbruksprognoser laget med nevrale nettverk. En prognosemodell baser på et nevralt nettverk ble laget for prisområdet, NO1. Shapley additive explanations ble brukt under utvikling av modellen for utvalg av predikatorer og for å debugge modellen. Metoden ble også brukt for å forstå enkelte prognoser som ble produsert under testing av den siste modellversjonen. Enkeltforklaringer av prognosene ble laget med metoden ved å bruke tidligere, sammenlignbare prognoser som bakgrunnsdata. Under utvikling av modellene, førte forklaringene til økt nøyaktighet og bedre forståelse av modellene. Forklaringene som ble laget under testing av den siste modellen var intuitive og ga ett inblikk i hva hva som førte til prognosen.

Resultatene fra oppgaven antyder at forklarbar intelligens kan brukes til å forstå og forbedre forbruksprognoser. Forklaringene er derimot avhengig av valg av bakgrunnsdata for å lage intuitive forklaringer. Det kreves derfor mer forskning om implikasjonene rundt valg av bakgrunnsdata før metoden kan bli tatt i bruk.

Contents

List of Figures

List of Tables

List of Acronyms

ACF	AutoCorrelation Function
Adam	Adaptive moment estimation
AdaGrad	Adaptive subGradient descent
AI	Artificial Intelligence
AROME	Application of Research to Operations at Mesoscale
Bagging	Bootstrap aggregating
CNN	Convolutional Neural Network
DeepLIFT	Deep Learning Important FeaTures
DSM	Demand Side Management
DSO	Distribution System Operator
DST	Daylight Saving Time
ELU	Exponential Linear Unit
ENTSO-E	European Network of Transmission System Operators for Electricity
kNN	k-Nearest-Neighbour

LIME Local Interpretable Model-agnostic Explanations

MAPE Mean Absolute Percentage Error

MET Norwegian Meteorological Institute

MEPS MetCoOp Ensemble Prediction System

MI Mutual Information

MIMO Multi-Input Multi-Output

ML Machine Learning

MLR Multiple Linear Regression

MLP MultiLayer Perceptron

MSE Mean Squared Error

Nadam Nesterov-accelerated adaptive moment estimation

NSGD Nesterov Stochastic Gradient Descent

NVE Norges Vassdrags- og Energidirektorat

NWP Numerical Weather Prediction

PACF Partial AutoCorrelation Function

PV PhotoVoltaic

ReLU Rectified Linear Unit

RES Renewable Energy Source

RMSProp Root Mean Square Propagation

RMSE Root Mean Squared Error

SeLU Scaled exponential Linear Unit

SHAP SHapley Additive exPlanations

TDS THREDDS Data Server

TSO Transmission System Operator

XAI Explainable Artificial Intelligence

XGBoost eXtreme Gradient Boosting

Chapter 1

Introduction

To ensure a stable power frequency in any interconnected AC power system, there has to be an instantaneous balance between power consumption, also called power demand or electrical load, and power production (supply). As a result of the Energy Act of 1990 in Norway, and similar deregulation of the power sector in other parts of Europe, the sale and purchase of power primarily takes place in power exchanges, so that supply and demand adhere to the principles of a free market. Electricity however, has a unique property as a commodity of needing to be consumed the very same instant it is produced. Meanwhile, trading of electrical energy occurs ahead of time (>1 hour), so that the *precise* consumption during the operating hour is unknown ahead of time. There is therefore a need for accurate electrical load forecasts for multiple market participants with various forecast horizons. The TSO ensures that the balance is kept through the use of balancing reserves, but has to procure these reserves days to weeks ahead of time (Statnett, 2021b); power producers optimize their generation schedules and unit commitment based on forecasts up until the operating hour (Gandhi et al., 2016); DSO and the TSO plan ahead for future expansions of the power system based on load forecasts months to years ahead; and power retailers need both short-term and long-term forecasts for providing their customers with electrical energy, and selling/buying for profit. Electrical load forecasting is therefore, and has been for a long time, an important forecasting problem in the energy domain (Hong, 2010).

The dynamics of the power system and consumer landscapes are changing. Penetration of RES in the power system is increasing (NVE, 2020), which puts strain on balancing due to their intermittent nature and lack of inertia contribution (Ørum et al., 2015). Consumers are participating more actively in the power market in a two-fold manner. Firstly, by participating in self-production of electrical energy as so-called "prosumers"; and secondly, via including smaller consumers in DSM schemes, where consumers are given incentive to adjust their load up or down depending on the need of the DSO (Petrican et al., 2018). Large-scale electrification (EVs, transport sector, petroleum industry) of the society has the potential of not only increasing consumption (Spilde et al., 2019), but also changing load consumption patterns. All of these are disrupting the conventional methods of maintaining power system stability which motivates development and integration of new technology. For instance, with EVs expected to make up large parts of Norwegian automotives within 2030, DSM of EVs has the potential of alleviating the power grid at high-load hours by shifting the load to low-load hours (Horne et al., 2020).

Another emerging technology which has seen increased use in electrical load forecasting the past years, is the field of AI, and more specifically ML (Debnath and Mourshed, 2018). These models, especially deep neural networks and ensemble models, are able to capture the patterns of electrical load with great accuracy (Baliyan et al., 2015; Tian et al., 2018; He, 2017). However, with increased model sophistication there are also greater difficulties in understanding the decisions and forecasts made by the forecasting model. Many AI models are so-called "black boxes", meaning that the underlying mathematical relations leading to the forecast are not readily comprehensible. While simpler models, like a MLR model, might be able to tell you that: "x caused y to increase by z", other models are not readily interpretable is this manner. Note that most models are in essence comprehensible, but their sheer size and complexity hinder the ability to understand what is going on. For high stake situations involving long-term forecasts, shareholders might require full transparency of the forecasting model (Hong, 2014). Trusting such a model put in deployment is difficult since there is no way to understand what might have lead to the forecast. Moreover, when this kind of model fails, there is no feedback as to what might have gone wrong, hence, trust in the model may be lost.

The response to this issue in other fields which use ML in its prediction models is the field of XAI. The aim of XAI is to provide explanations for the model's predictions and to "look inside" the black box. So far, the use and importance of XAI has been explored in several sectors where critical decision-making based on ML models has to be justified (e.g., medicine (Lundberg et al., 2020), legal sector (Doshi-Velez et al., 2017) and national defense-systems (Turek, 2018)). However, based on extensive literature searches, including multiple searches on Google Scholar, ResearchGate, IEEE Xplore, ScienceDirect and arXiv using different combinations of keywords like "explainable", "XAI", "energy forecasting", "forecasting", "load", "solar", "wind" etc., there has been little research concerning the use of XAI in the energy and load forecasting fields.

Among the few studies found, Grimaldo and Novak (2020) used visual analytics in the shape of dashboards to visualize the results of forecasted local energy demand and supply, where the forecast was based on a kNN model, and the dashboard showed demand and supply of similar days to explain the predictions. They found that the end-users were able to understand individual predictions made by the kNN algorithm, without any background knowledge of the model. Arjunan et al. (2020) used SHAP to determine factors with the largest effect on building energy performance benchmarks and found that SHAP was useful for visualizing the predictions even for non-technical users. Kuzlu et al. (2020) investigated three different XAI frameworks for PV power forecasting to uncover the most influencing features of an XGBoost model. They found that all three XAI frameworks were useful for feature selection, and omitted the least important features without much loss in performance. Lee et al. (2020) used XAI techniques to find the most influencing features in a load forecasting model using XGBoost. Using this information in the feature engineering process, they were able to improve the performance of the model's future iterations. Lastly, Ilic et al. (2020) used an explainable boosted linear regression model for load forecasting and extracted feature importances in order to extract the most influencing features.

Norwegian case studies of electrical load forecasting using ML are also limited. In his master thesis on short-term load forecasting (20 to 43 hours ahead) from NTNU, Tyvold (2018) developed multiple different regional ML forecasting models for the DSO in Nord-Trøndelag,

NTE. He found that his models outperformed NTE's proprietary model while also being more efficient. Statnett's data science department published a brief case study on their blog page of using ML for online day-ahead load forecasting of all the Norwegian price regions (Presthus, 2018). They found that the new models outperformed their proprietary solutions in every price region except NO1. They also noted the potential use of XAI to make forecasts more accessible to their balancing operators and to provide insight of the model.

The above case studies used black box AI models to outperform existing models, and pointed out the impressive accuracy of such models. However, investigation of how XAI can be used to explain Norwegian load forecasts were not part of their scopes. In fact, no works in the literature for this type of problem in the Norwegian power system were found. The following thesis therefore aims to fill this gap, by serving as a first exploratory work in the use of XAI for Norwegian load forecasts. In other words, how can XAI tools be leveraged to interpret and improve load forecasting models?

To answer this question, a case study was made for day-ahead electrical load forecasting of the NO1 price region in Norway. The particular contributions of the thesis are three-fold:

(1) Development of a day-ahead electrical load forecasting model for the NO1 price region

(2) Showcasing how an XAI framework can be used during model development to improve and understand the forecasting model

(3) Interpreting electrical load forecasts made by a model in deployment using XAI visualizations

The thesis structure is as follows: chapter 2 covers the theoretical background; chapter 3 investigates the data sets of the case study; chapter 4 describes experimental setup, models and tests in detail; chapter 5 contains the results and a discussion of them; and lastly, in chapter 6 some concluding remarks, standing issues and ways forward are given.

Chapter 2

Background theory

The following chapter covers the theoretical foundations needed to understand and develop the models presented in chapter 4. There are four fundamental areas which will be investigated: time series and selected statistics; AI and supervised learning; neural networks and optimization; and XAI.

2.1 Time series and statistics

A time series captures the development of a target variable over a discrete set of time points. It is represented by chronological data, where each data point is the target's value at a particular time. The points are typically spaced apart with constant intervals, which is referred to as the resolution of the time series. E.g., there may be an observed value for every five minutes, 30 minutes or every hour. As an example, figure 2.1 shows a time series with an hourly resolution. In the following sections, the observed value of the target at some time t will be denoted as $y^{(t)}$, and the value of one time-step ahead as $y^{(t+1)}$. Depending on the time-span and the target involved, characteristics of the target variable like trend or seasonality may be evident. Trend is the general direction (up or down) of the time series observed over a long period of time, usually years, and may be identified through a moving-average of the time series. A time series with seasonal components on the other hand, may be observed sub-daily, daily, weekly or any other fixed time period. Seasonality

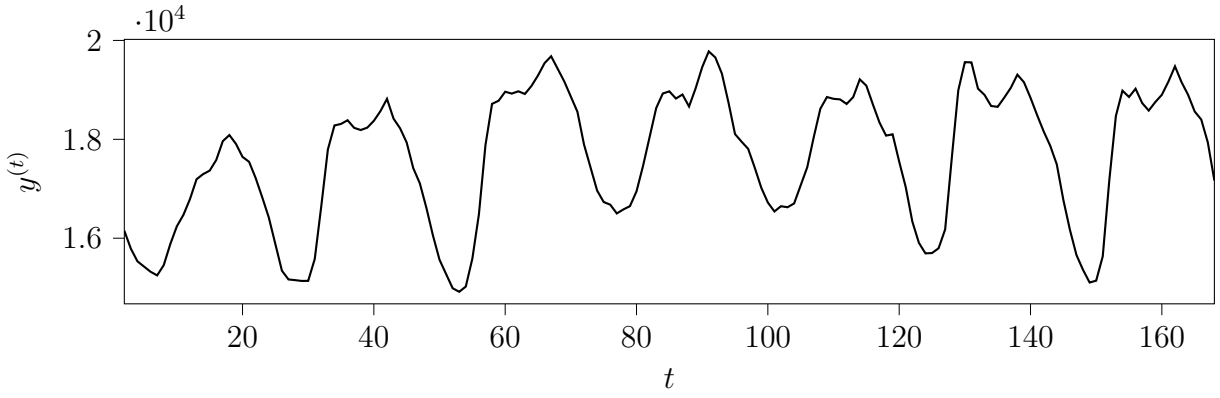


Figure 2.1: Example of a time series.

appears as peaks and troughs that are consistently spaced apart and repeat themselves in a periodic fashion throughout the time series. This can be seen in figure 2.1, which has a daily seasonality (i.e., peaks and troughs are spaced apart by about twenty-four points of one hour each). If a time series contains seasonality, trend or if the variance is changing with time, the time series is considered non-stationary, i.e., time-dependent. If the time series is seasonal, the changing variance can be observed as an increase in the seasonal peaks over time.

While conventional time series modelling approaches require the time series to be stationary (Hyndman and Athanasopoulos, 2018), there is debate whether adjustment and/or transformation of the time series is necessary for modelling approaches that use approaches deemed as AI (Claveria et al., 2017). Nevertheless, it may be worthwhile to investigate different combinations of pre-processing of the time series and the performance impact on the problem at hand (Bianchi et al., 2017; Makridakis et al., 2018). In the following sections, frequently applied methods for statistical analysis of a time series, and commonly applied transformations will be investigated. Lastly, some strategies of forecasting future values of a time series will be covered.

2.1.1 Autocorrelation

Autocorrelation gives a measure of the correlation between the target variable and *lagged* versions of its self. That is, a time series that is a copy of the target, shifted in time so that it lags the target. This is useful in order to determine which lags to include in a

forecasting model. It is also a way of identifying seasonal and trend aspects of a time series. The autocorrelation coefficient is the Pearsons's correlation coefficient between the target variable and its lagged values. The coefficient therefore measures how much of the variable's current value is accounted for by its past ones (Hyndman and Athanasopoulos, 2018). For a lag of k time steps, the coefficient is given by

$$r_k = Cor(y^{(t+k)}, y^{(t)}) = \frac{\sum_{t=1}^{N-k}(y^{(t)} - \bar{y})(y^{(t+k)} - \bar{y})}{\sum_{t=1}^{N}(y^{(t)} - \bar{y})^2}, \tag{2.1}$$

where $Cor(\cdot)$ denotes the Pearson's correlation function and $\bar{y}$ is the average of the target variable across the time series. Similarly to correlation between two different variables, the autocorrelation coefficient takes on the range of values $r_k \in [-1, 1]$, where the limits denote perfect negative and positive correlation respectively. I.e. situations where the past value of the target completely predicts the current value.

The autocorrelation coefficient may be plot as a function of the lag, which is often referred to as the autocorrelation function (ACF). For seasonal time series, spikes in the ACF plot will be apparent at multiples of the seasonal period (e.g., at lags $0, 24, 48, \ldots$ for hourly data with daily seasonal patterns). Additionally, if the time series has a trend component, the autocorrelation will decay with increasing lag (Hyndman and Athanasopoulos, 2018). It is common to also include a confidence interval (usually the 95%) when plotting autocorrelation as a function of lag. This is done in order to determine the significance of the autocorrelation coefficient at each lag, and to rule out potential noise.

If $y^{(t)}$ and $y^{(t+1)}$ are correlated, then it follows that $y^{(t+2)}$ will be correlated to $y^{(t+1)}$, hence also correlated to $y^{(t)}$. In other words, the correlation between $y^{(t)}$ and $y^{(t+k)}$ may be attributed to intermediary lags. In order to consider the effect of the kth lag alone, the partial autocorrelation function (PACF) may be used instead (Hyndman and Athanasopoulos, 2018). The PACF is similar to the ACF, except that the effects of intermediary lags have been removed for the coefficient of a particular lag.

2.1.2 Mutual information

Mutual information (MI) gives a measure of the mutual dependence between two random variables x and y. It may be formulated as

$$I(y;x) = H(y) - H(y|x), \tag{2.2}$$

where H is the entropy (uncertainty). Hence, the MI may be interpreted as the average reduction of uncertainty in y from knowing x (MacKay, 2003, p. 139). MI has a bound of $I \in [0, \infty$, where zero means that the two variables are independent and $I \neq 0$ implies that some information can be extracted from each other. Moreove, MI has a symmetry property in which $I(y;x) = I(x;y)$.

2.1.3 Normalization

Normalization, or rescaling of a variable involves transforming the variable to some preferred distribution or range. In the context of several inherently different variables, normalization is a way of transforming the variables to comparable scales. The two most commonly used normalization methods are min-max normalization and Z-score normalization (also called standardization). Min-max normalization refers to rescaling the variable to some range $[a, b]$ (usually $[0, 1]$ or $[-1, 1]$). Rescaling some variable $x \in [x_{min}, x_{max}]$ to $x' \in [0, 1]$ is given by

$$x' = \frac{x - x_{min}}{x_{max} - x_{min}}. \tag{2.3}$$

Standardization involves transformation of the variable to a normal distribution with zero mean and unit variance. This transformation may be given by

$$x' = \frac{x - \mu}{\sigma}, \tag{2.4}$$

where μ and σ are the mean and standard deviation of the variable, respectively.

2.1.4 Time series forecasting methods

A time series forecasting scheme may be described by the forecasting horizon, interval and its resolution (Montgomery et al., 2008, p. 5). In other words, how far ahead in time to forecast, how often to produce these forecasts and how much time there is between each output. The forecasting interval is often also referred to as the cycle of the forecasting scheme. In particular, let t denote the time of forecast (termin or reference time), and using the K previously observed values of the target and predicting the H next. Later at a time $t+I$, where I is the forecasting interval, a new forecast is made. The first forecast value may be at $t+1$, but it can also be after some delay, also known as lead time, from the termin time $t' = t + D$ (e.g., day-ahead forecasting where $D = 24$).

The multitude of forecasting methods may be separated into those belonging to single step-ahead ($H = 1$) or multiple steps-ahead ($H > 1$) methods (Brownlee, 2018). A single time step, as the name suggests, fits a function f to predict the target value $\hat{y}^{(t+1)}$ one step ahead, and may be modelled as

$$\hat{y}^{(t+1)} = f(y^{(t)}, y^{(t-1)}, \ldots, y^{(t-K+1)}). \tag{2.5}$$

Multiple steps-ahead forecasting strategies can be divided into five distinct methodologies (Ben Taieb et al., 2012). These are:

(1) Recursive models

(2) Direct models

(3) Multi-input multi-output models (MIMO)

(4) Combination of direct and recursive models

(5) Combination of direct and MIMO models

Recursive models utilize single-step-ahead forecasts in an iterative fashion, where the predicted value is fed back to function as historical data for the forecast. These models may

however be prone to large errors with long forecasting horizons or poor single-step performance, due to propagation of the error further down the chain of predictions.

Direct models fit an independent function for each respective time step, so that H different forecasts are made. This can be described similarly to equation 2.5, where

$$\hat{y}^{(t+h)} = f_h(y^{(t)}, y^{(t-1)}, \ldots, y^{(t-K+1)}), \tag{2.6}$$

and f_h denotes the function fit to h time steps ahead. The drawback of these models is that they assume independence between the time steps, which neglects any interaction effects that may exist. They are also more computationally expensive than their counterparts.

MIMO models fit a vector-spaced function that generates multiple outputs at once, thereby avoiding the assumption of independence. These models are also relatively efficient, since only one model is required, but are also less flexible as a result. The MIMO forecasting strategy may be formulated as

$$\begin{bmatrix} \hat{y}^{(t+H)} & \hat{y}^{(t+H-1)} & \ldots & \hat{y}^{(t+1)} \end{bmatrix}^T = f(y^{(t)}, y^{(t-1)}, \ldots, y^{(t-K+1)}), \tag{2.7}$$

where $f : \mathbb{R}^K \to \mathbb{R}^H$.

Combinations of the aforementioned forecasting strategies, combine aspects of the methods in order to alleviate the weaknesses inherent in a standalone method. E.g., Ben Taieb et al. (2012) proposed a method using a combination of direct and MIMO. The proposed strategy outperformed the other methods, except the MIMO model which provided similar forecasting accuracy.

All of the above are endogenous forecasting strategies. That is, they only use the information provided by the previous values of the target variable itself. Exogenous variables may be included to any of the above methods to further improve the accuracy of the forecast. These are variables that may help describe the behaviour of the target variable. This means historical or future values of other time series variables, and categorical variables such as day of the week, holidays and month of the year. For instance, in the case of electrical

load forecasts, the influence of weather variables on consumption patterns is well established (Hong, 2010). Acquiring numerical weather predictions (NWP) and extracting suitable time series of temperature, wind, humidity, etc., may therefore provide important additional information to the forecasting model (Hong et al., 2015).

2.2 Artificial intelligence

Artificial intelligence is a broad term that usually refers to the more tangible fields of machine learning (ML) and deep learning. These two fields involve the use of large amounts of data to *learn* some process through mathematical modelling and optimization. Within the field of ML, it is common to distinguish between the methods belonging to supervised, unsupervised or reinforcement learning. A supervised learning problem may be described either as a regression or classification problem, based on whether the targets have continuous or discrete values, respectively (James et al., 2013, p. 28). In the context of power load, it can in theory be any real number $P \in \mathbb{R}$. Additionally, modelling the forecasting problem explicitly as seen in section 2.1.4, means a supervised learning algorithm is the most suitable. The following sections therefore put particular emphasis on regression based supervised learning.

2.2.1 Supervised learning

In a supervised learning problem an attempt is made to accurately predict explicitly given targets in a data set, based on the corresponding features (also called inputs, predictors, independent variables or similar). In particular given a pair of feature and target matrices

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}^{(0)} \\ \boldsymbol{x}^{(1)} \\ \vdots \\ \boldsymbol{x}^{(N-1)} \end{bmatrix} \quad \text{and} \quad \boldsymbol{Y} = \begin{bmatrix} \boldsymbol{y}^{(0)} \\ \boldsymbol{y}^{(1)} \\ \vdots \\ \boldsymbol{y}^{(N-1)} \end{bmatrix}, \tag{2.8}$$

a function is fit to map features to targets given by

$$\hat{\boldsymbol{y}} = f(\boldsymbol{x}), \tag{2.9}$$

where $f : \mathbb{R}^n \to \mathbb{R}^m$, N are the number of data points, n are the number of features and m are the number of targets; $\boldsymbol{x}^{(i)} = \begin{bmatrix} x_1^{(i)} & x_2^{(i)} & \cdots & x_n^{(i)} \end{bmatrix}$ and $\boldsymbol{y}^{(i)} = \begin{bmatrix} y_1^{(i)} & y_2^{(i)} & \cdots & y_m^{(i)} \end{bmatrix}$ make up one instance, or sample, of feature and target vectors; and $\hat{\boldsymbol{y}}^{(i)}$ is the corresponding vector of predicted values for the instance (Bishop, 2008, p. 3).

For parametric supervised learning methods, the shape of the underlying function is assumed to be known and is determined through a set of parameters, denoted as θ. A parametric supervised learning problem may then be formulated as an optimization problem, where we want to minimize the cumulative losses of the model with respect to the choice of parameters. A commonly chosen loss function, also often named cost function or error function, in regression problems is the mean squared error (MSE). MSE is a metric of the mean squared deviations from the actual target values (James et al., 2013, p. 21). Drawing a training set, denoted as $\mathcal{D}$ from the pair of feature and target matrices, the MSE for the training loss may be given as

$$L(\boldsymbol{\theta}) = \frac{1}{N_{\mathcal{D}}} \sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathcal{D}} ||\hat{\boldsymbol{y}}(\boldsymbol{x};\boldsymbol{\theta}) - \boldsymbol{y}||^2, \tag{2.10}$$

where $N_{\mathcal{D}}$ are the number of samples in the training set, hence the training set may be described as $\mathcal{D} = \{(\boldsymbol{x}^{(0)}, \boldsymbol{y}^{(0)}), (\boldsymbol{x}^{(1)}, \boldsymbol{y}^{(1)}), \ldots, (\boldsymbol{x}^{(N_{\mathcal{D}}-1)}, \boldsymbol{y}^{(N_{\mathcal{D}}-1)})\}$; $||\cdot||$ denotes the L^2 norm; and the semicolon notation $\boldsymbol{x};\boldsymbol{\theta}$ refers to the parameters of the supervised learning model, and that these are fixed during the calculation of the loss (Bishop, 2008, p. 233).

The analytical solution of minimizing the error function can be given in closed form for simple models, e.g., linear regression (see section 2.2.2). For other models, numerical methods may need to be applied to reach a solution through iterative algorithms such as a gradient descent. In the case of non-convex optimization problems, numerical methods are not guaranteed to move towards the global minimum, but instead a local minimum. However, in the context of ML problems these usually have losses that are small enough to be considered satisfactory (Goodfellow et al., 2016, p. 282-285). In a gradient descent algorithm, the idea

is to keep descending down the loss surface by updating the parameters in steps opposite to the direction of the gradient of the loss function. An update of the parameters from one iteration of gradient descent to the next is given by

$$\boldsymbol{\theta}_{\tau+1} = \boldsymbol{\theta}_{\tau} - \eta \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}_{\tau}), \tag{2.11}$$

where η is the learning rate and τ is the current iteration of the algorithm (Bishop, 2008, p. 144). The gradient gives the steepest direction of the error surface given an infinitesimally small change in parameters, hence the learning rate determines the actual step-size of each update. It is therefore a crucial hyper-parameter to tune, both in terms of reaching convergence and having an acceptable total runtime of the optimization.

A drawback of the standard gradient descent algorithm is that for every iteration, the gradient of the error function has to be calculated across the entire set of predicted and target values (as seen in equation 2.10). For models that are trained on very large data sets while also having millions of parameters, the standard gradient descent algorithm is infeasible due to computational complexity (Goodfellow et al., 2016, p. 149). A variation of the gradient descent, known as the stochastic gradient descent (SGD), estimates the gradient by using a randomly picked instance of the training set instead. I.e., pick a random instance from the training set $\mathcal{D}$, so that the next update to the parameters is given by

$$\boldsymbol{\theta}^{\tau+1} = \boldsymbol{\theta}^{\tau} - \eta \nabla_{\boldsymbol{\theta}} \ell(\boldsymbol{\theta}^{\tau}; \boldsymbol{x}, \boldsymbol{y}), \tag{2.12}$$

where $(x, y) \in \mathcal{D}$; and ℓ refers to the loss of that single instance (Gu et al., 2018, p. 15). Another more widely used variation, named mini-batch SGD, splits the training set into batches of randomly picked instances from the training set. The actual gradient is then estimated using the average of the gradients in a batch. The number of random samples in each mini-batch is referred to as the batch size of the algorithm. When all mini-batches have been used to update the parameters, i.e., all instances in the training set have been used, this is know as the end of one epoch. E.g., given a batch size of N_b, $\frac{N_{\mathcal{D}}}{N_b}$ iterations will make up one epoch.

It is important to note that although the training loss minimum may eventually be reached, this is generally not the goal. In order to perform well on unseen data, i.e., instances outside of the training set, the model also needs to have a sufficiently low generalization loss (Goodfellow et al., 2016, p. 290). By solely minimizing the training loss, the parameters will be prone to over-fit to the training data, and in turn perform poorly on unseen data. In practice, equation 2.10 is therefore used as a surrogate loss function. Particularly, gradient descent with the surrogate loss function keeps running as long as the loss on a validation set also keeps decreasing (Goodfellow et al., 2016, p. 274). The idea is that the validation data is representative of the underlying distribution. Hence, if the training data also represents the underlying distribution, both the validation loss and training loss should decrease in unison. When the validation loss stops improving, but the training loss keeps decreasing, the model is beginning to bias the training set and the optimization algorithm should stop. This is known as early stopping because instead of stopping based on the maximum number of epochs or when $\nabla_{\boldsymbol{\theta}} L = \mathbf{0}$, the algorithm stops when the validation loss stops improving (i.e., has not improved by more than ϵ in n epochs) (Goodfellow et al., 2016, p. 240).

2.2.2 Multiple linear regression

Following the convention from equation 2.9, perhaps the simplest relationships between features and targets which can be assumed, is a linear one. This relationship is given by a multiple linear regression (MLR) model, where a prediction $\hat{y}_k$ is given by

$$\hat{y}_k = \beta_{k1} x_1 + \beta_{k2} x_2 + \cdots + \beta_{kn} x_n. \tag{2.13}$$

In equation 2.13, β_{kj} is the regression coefficient of the model pertaining to the kth target and jth feature. Given the feature matrix $\boldsymbol{X}$ from equation 2.8, the predictions of the MLR model are given by

$$\hat{\boldsymbol{y}}_k = \boldsymbol{X}\boldsymbol{\beta}_k \tag{2.14}$$

where $\boldsymbol{\beta}_k = \begin{bmatrix} \beta_{k1} & \beta_{k2} & \cdots & \beta_{kn} \end{bmatrix}^T$ and $\hat{\boldsymbol{y}}_k = \begin{bmatrix} \hat{y}_k^{(1)} & \hat{y}_k^{(2)} & \cdots & \hat{y}_k^{(N)} \end{bmatrix}^T$. Equation 2.14 has no unique solution, hence the solution space has to be given in the form of an optimization

problem. One choice is to solve the system through ordinary least squares. Then the optimization problem may be formulated as

$$arg\min_{\boldsymbol{\beta}_k} L(\boldsymbol{\beta}_k) = arg\min_{\boldsymbol{\beta}_k} \sum_{i=1}^{N} |y_k^{(i)} - \hat{y}_k^{(i)}|^2 = ||\boldsymbol{y}_k - \boldsymbol{X}\boldsymbol{\beta}_k^T||^2 \tag{2.15}$$

It can be shown (see Hastie et al. (2008, p. 12)) that the unique solution to this system is given by

$$\boldsymbol{\beta}_k = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}\boldsymbol{y_k}. \tag{2.16}$$

2.3 Artificial neural networks

Artificial neural networks, or just neural networks, have risen in popularity since the early 2000s with increased availability of processing power and large amounts of data (Goodfellow et al., 2016, p. 222). They have been proven to be universal non-linear function approximators under the right circumstances, i.e., they can approximate any given function (Nielsen, 2015), and are therefore suitable for prediction of complex processes. With many fully implemented frameworks available open source, practitioners can easily get started with neural networks having little background knowledge. However, when fine-tuning the network architecture to the problem at hand, it is useful to be aware of the underlying mechanisms to gain an intuitive understanding of their behaviour.

2.3.1 Multilayer perceptron

Perhaps the most widely used type of neural network is the multilayer perceptron (MLP), often called a feed-forward neural network due to its structure (Bishop, 2008, p. 226). The MLP is made up of a number of layers, consisting of the input and output layers and a number of hidden layers between the two. Each hidden layer is made up of an arbitrary number of hidden units, determining the number of outputs from that hidden layer. Each hidden unit performs a weighted linear combinations of its inputs, where the result is fed through an activation function before being sent to the next layer. These functions enable the network to form non-linear representations and are important to the overall network

architecture.

Consider for instance an MLP with inputs, outputs and two hidden layers. The linear combination for one hidden unit in the first hidden layer is given by

$$z_i^{[1]} = b_i^{[1]} + \sum_{j=1}^{n} w_{ij}^{[1]} x_j, \tag{2.17}$$

where $z_i^{[1]}$ is the weighted input of the ith hidden unit in the first hidden layer; $b_i^{[1]}$ is the bias; x_j is the jth input; and $w_{ij}^{[1]}$ is the weight between the ith unit and jth input. Usually, only one distinct activation function is used for each layer. The output of the hidden unit is then given by feeding the weighted input through the activation function of the layer. This may be given by

$$a_i^{[1]} = \phi^{[1]}(z_i^{[1]}), \tag{2.18}$$

where $a_i^{[1]}$ is the activation of the ith unit in the first hidden layer, and $\phi^{[1]}$ is the activation function of the first hidden layer.

This procedure is then continued in the next layer, where the activations of the previous layer ($\boldsymbol{a}^{[1]}$) are the inputs of each hidden unit. After feeding the activations from the first hidden layer through the second, the activations from the second hidden layer are linearly combined to form the output layer weighted inputs, given by

$$z_k^{[3]} = b_k^{[3]} + \sum_{j=1}^{n^{[2]}} w_{kj}^{[3]} a_j^{[2]}, \tag{2.19}$$

where $z_k^{[3]}$ denotes the weighted input of the kth output unit (Bishop, 2008, p. 228). Finally these are fed through the output activation function to give the model outputs, which may be formulated as

$$y_k = \phi^{[3]}(z_k^{[3]}). \tag{2.20}$$

The above MLP is illustrated in figure 2.2. It has n inputs; m outputs; and $n^{[1]}$ and $n^{[2]}$ hidden units in the first and second hidden layers, respectively. Since each unit is connected to all of its preceding and succeeding units in every layer, this type of MLP is usually referred

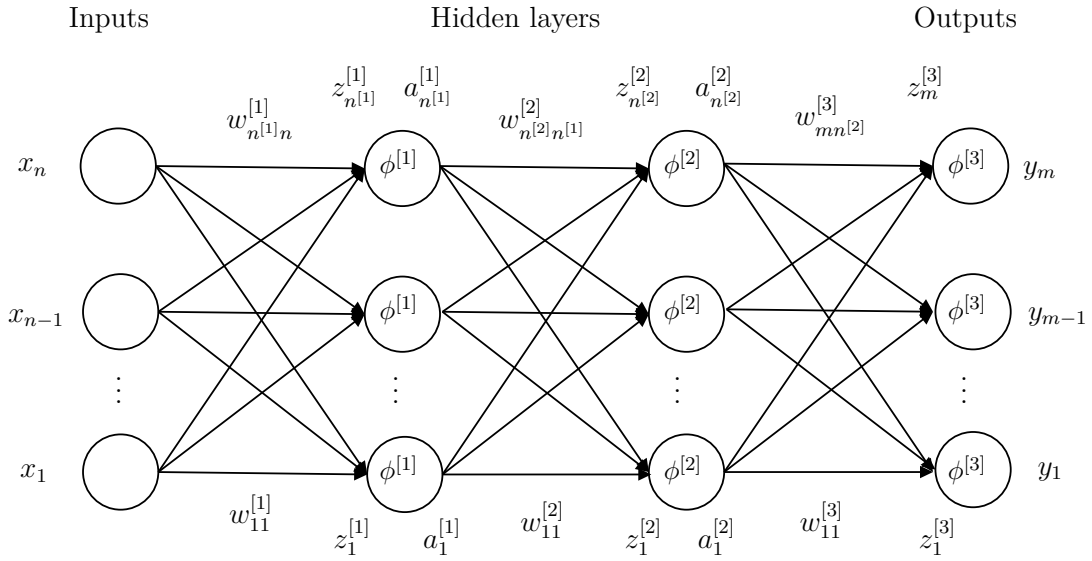


Figure 2.2: Illustration of a multilayer perceptron with two hidden layers. Adapted from Bishop (2008, p. 228).

to as a fully connected neural network (Grosse, 2018). The description of any MLP may be given in vectorized notation by denoting the biases of the lth layer as $\boldsymbol{b}^{[l]}$; the matrix containing the weights for the lth layer as $\boldsymbol{W}^{[l]}$, where the notation $w^{[l]}_{ij}$ corresponds to the ith row and jth column of the matrix; and the vector of activations as $\boldsymbol{a}^{[l]}$ (Nielsen, 2015). The equations describing any fully connected MLP are then given by

$$\boldsymbol{a}^{[1]} = \phi^{[1]}(\boldsymbol{W}^{[1]}\boldsymbol{x} + \boldsymbol{b}^{[l]}) \tag{2.21}$$

$$\boldsymbol{a}^{[l]} = \phi^{[l]}(\boldsymbol{W}^{[l]}\boldsymbol{a}^{[l-1]} + \boldsymbol{b}^{[l]}) \tag{2.22}$$

$$\boldsymbol{y} = \phi^{[L]}(\boldsymbol{W}^{[L]}\boldsymbol{a}^{[L-1]} + \boldsymbol{b}^{[L-1]}), \tag{2.23}$$

where L is the number of hidden layers plus the output layer in the network; $\boldsymbol{W}^{[L]}$ denotes the weights of the output layer; and ϕ is the element-wise activation function (Grosse, 2018). The dimensionality of the activations in a hidden layer is $\boldsymbol{a}^{[l]} \in \mathbb{R}^{n^{[l]}}$. Hence, in the case of a fully connected network, the weights of that layer will be $\boldsymbol{W}^{[l]} \in \mathbb{R}^{n^{[l]} \times n^{[l-1]}}$, where $n^{[l]}$ denotes the number of hidden units in layer l. Notice how the outputs of each layer are functions of the outputs of preceding layers. This is known as forward propagation, and will be a part of the discussion of the backpropagation algorithm in the next chapter.

2.3.2 Backpropagation

Recall that a supervised learning model may be trained through using the SGD algorithm or its extensions (equation 2.12). These algorithms however, do not include how to find the gradient for a particular iteration. For models that are simple functions of the parameters, e.g., linear regression, determining the gradient of the loss function does not pose a great amount of difficulty. However, for neural network which are composed of several non-linear layers and many hidden units, calculating the gradient is computationally expensive and was for a long time a barrier for the wide-spread use of neural networks in many problem domains (Nielsen, 2015; Goodfellow et al., 2016, p. 200).

The solution to this problem was the development of backpropagation (Linnainmaa, 1970). The backpropagation algorithm provides an efficient means for calculating the gradient based on propagating the loss backwards in the network. This is done in order to evaluate the partial derivatives of the loss with respect to all the weights and biases from the lowest layers (the ones the furthest back from the output layer) to the top layers. The advantage of backpropagation, is that it does this quite efficiently for networks with many parameters by use of the chain rule (Nielsen, 2015; Goodfellow et al., 2016, p. 206).

Full derivation of the backpropagation equations is shown in appendix A. The vectorized equations are given by

$$\boldsymbol{\delta}^{[L]} = \nabla_{\hat{\boldsymbol{y}}}\ell \odot \phi'(\boldsymbol{z}^{[L]}) \tag{2.24}$$

$$\boldsymbol{\delta}^{[l]} = (\boldsymbol{W}^{[l+1]})^T\boldsymbol{\delta}^{\boldsymbol{l+1}} \odot \phi'^{[l]}(\boldsymbol{z}^{[l]}) \tag{2.25}$$

$$\nabla_{\boldsymbol{W}^{[l]}}\ell = \boldsymbol{\delta}^{[l]}(\boldsymbol{a}^{[l-1]})^T \tag{2.26}$$

$$\nabla_{\boldsymbol{b}^{[l]}}\ell = \boldsymbol{\delta}^{[l]}, \tag{2.27}$$

where $\ell = \ell(\boldsymbol{\theta}^\tau; \boldsymbol{x}, \boldsymbol{y})$ is the individual loss of some mini-batch; $\boldsymbol{\delta}^{[l]} = \nabla_{\boldsymbol{z}^{[L]}}\ell$ is the error vector of the layer l; $\nabla_{\boldsymbol{W}^{[l]}}\ell$ is the gradient of ℓ with respect to the weights in layer l; and $\nabla_{\boldsymbol{b}^{[l]}}\ell$ is the gradient of ℓ with respect to the biases in layer l.

By use of equations 2.24-2.27 the gradients of the losses with respect to every layer can be

found. Particularly, by starting at the output layer (equation 2.24) and moving backwards in the network (equation 2.25), the gradients of all the parameters (equation 2.26 and 2.27) can be calculated in each layer by the use of programming libraries with fast vectorized calculations (Nielsen, 2015).

One update to the model parameters is then made in the following manner: (1) for each instance in a mini-batch, a forward pass (equations 2.21-2.23) is made in order to determine all the current values of the network; (2) the gradients for all layers are found through the use of the backpropagation equations (3) the average of the gradients over the mini-batch is then calculated and the weights and biases are updated. The average of the gradient of the total loss L may be given by an average over the individual gradients

$$\nabla_{\boldsymbol{\theta}} L = \frac{1}{N_b} \sum_{\boldsymbol{x},\boldsymbol{y} \in \mathcal{B}} \nabla_{\boldsymbol{\theta}} \ell(\boldsymbol{\theta}^{\tau}; \boldsymbol{x}, \boldsymbol{y}) \tag{2.28}$$

where N_b is the batch size and $\mathcal{B}$ is the mini-batch of the current iteration. Equation 2.28 may then be used in conjunction with equation 2.12 as before. Note that $N_b = 1$ is the special case of using standard SGD optimization, i.e., there is only one instance to be evaluated per parameter update. The backpropagation equations can also be modified to calculate the average gradient of a batch directly by implementing the algorithm with matrix based equations (Nielsen, 2015).

2.3.3 Activation functions

The activation functions in a neural network are essential, since they allow the network to learn non-linear representations of its inputs (Goodfellow et al., 2016, p. 136). The choice of the output layer activation function is usually based on the choice of cost function. In particular, notice from equations 2.21-2.23 that if ϕ is a linear transformation, then the MLP would essentially be a linear model. In regression problems that use MSE as the cost function, this is usually the choice of the output activation function. Particularly, the output activation function is set to an identity function (Goodfellow et al., 2016, p. 177). Using a different cost function (such as cross-entropy) may warrant the use of other output activation functions, but will not be covered since MSE is usually selected for regression problems.

For the hidden layers in the network, there are no rules for choosing activation functions. Rather, the choice is a part of the hyper-parameter optimization procedure and is chosen through trial and error by observing the validation set error (Goodfellow et al., 2016, p. 178). There are however some commonly chosen activation functions for the hidden units, and some default recommendations. A rectified linear unit (ReLU) defines the activation function as

$$\phi(z) = max(0, z), \tag{2.29}$$

i.e., it is a piece-wise linear function propagating only positive linear combinations. The ReLU is usually recommended as the default hidden unit for MLPs, due to easy optimization with gradient-based algorithms and their ability to generalize (Goodfellow et al., 2016, p. 170). In particular, the ReLU may alleviate the problem of vanishing gradients, which is the phenomenon where earlier layers of a network learn much slower than the topmost layers.

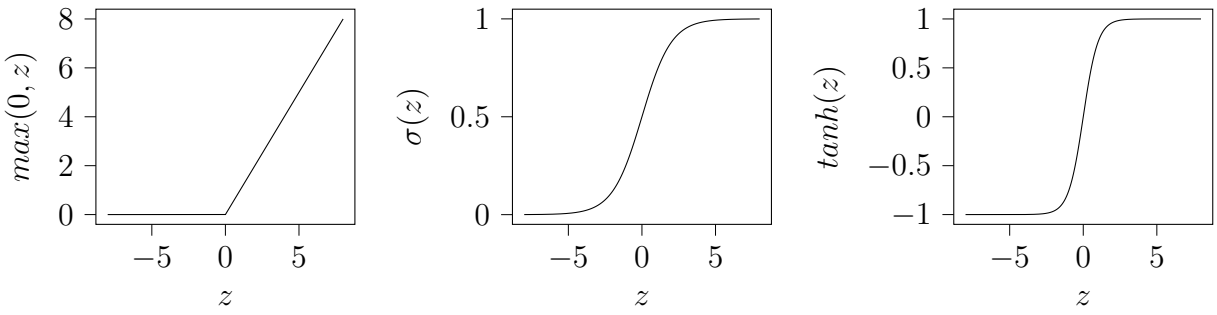
The logistic sigmoid activation function may be described by

$$\phi(z) = \sigma(z) = \frac{1}{1 + e^{-z}}. \tag{2.30}$$

As seen from figure 2.3, the sigmoidal function saturates when its input is small or large, which may pose a problem for gradient-based optimization. I.e., notice that $\sigma'(z) = 0$ for small and large input values, which in turn stops learning due to small gradients (e.g., see equation A.6). For this reason, the use of sigmoidal hidden units in MLPs is usually discouraged (Goodfellow et al., 2016, p. 191). Instead, the hyperbolic tangent function which is given by

$$\phi(z) = tanh(z), \tag{2.31}$$

is recommended in place of the sigmoid. This is partly due to the fact that the hyperbolic tangent acts like an identity function for values close to zero. I.e., that $tanh(z) = z$ for small values (Goodfellow et al., 2016, p. 192). By standardizing (see section 2.1.3) so that features are centered around zero, this can often be the case. The shapes of equations 2.29, 2.31 and 2.30 are shown in figure 2.3.



(a) Rectified linear (b) Logistic sigmoid (c) Hyperbolic tangent

Figure 2.3: The shape of some classically used activation functions. (a), (b) and (c) show the rectified linear, logistic sigmoid and hyperbolic tangent activation functions respectively.

Other choices of activation functions for hidden units include softmax, radial basis function, softplus and variations of the ReLU. Some of these are difficult to optimize due to saturation or have shown to be inferior to the three above in most cases (Goodfellow et al., 2016, p. 193). Variations of the ReLU, however, may perform well compared to the standard ReLU, sigmoid and hyperbolic tangent in certain situations. For instance, the leaky ReLU, parametrized ReLU, exponential linear units (ELU) and scaled exponential linear units (SELU), all have negative activations in contrast to the ReLU (Clevert et al., 2016). Particularly, a leaky ReLU defines the activation function as

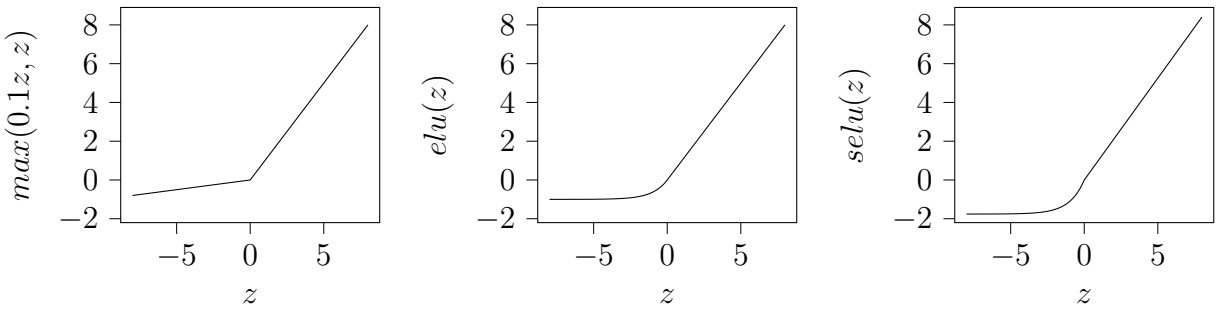
$$\phi(z) = max(\zeta z, z), \tag{2.32}$$

where ζ is a hyper-parameter which adjusts the slope for $z < 0$; ELU is given by

$$\phi(z) = elu(z) = \begin{cases} \alpha(e^z - 1) & x < 0 \\ z & x \geq 0 \end{cases}, \tag{2.33}$$

where α is a hyper-parameter usually set to 1; while SELU is given by

$$\phi(z) = selu(z) = \lambda \begin{cases} \alpha(e^z - 1) & x < 0 \\ z & x \geq 0 \end{cases}, \tag{2.34}$$



(a) Leaky ReLU (b) Exponential linear (c) Scaled exponential linear

Figure 2.4: Variations of the ReLU. (a), (b) and (c) show the leaky ReLU, ELU and SELU activation functions respectively.

where $\alpha = 1.67326324$ and $\lambda = 1.05070098$ (Chollet et al., 2015). Equations 2.32, 2.33 and 2.34 are shown for $z \in [-8, 8]$ in figure 2.4.

The negative values of leaky ReLU, ELU and SELU mean that the derivatives of these activation functions are non-zero for negative values, which may help during optimization. Particularly to alleviate the vanishing gradient issue (Chollet et al., 2015). However, note that there is no universally agreed upon "best" activation function. The recommendations are mostly based on empirical evidence, where ReLUs, its variations and hyperbolic tangent units usually perform well.

2.3.4 Convolutional neural networks

Another type of widely used neural network is the convolutional neural network (CNN). Although CNNs have been popularly used for two-dimensional applications (such as images (Goodfellow et al., 2016, p. 366)), the convolutions that are made on the input data may be generalized to arrays of arbitrary dimensions (Dumoulin and Visin, 2016). These multi-dimensional arrays are known as tensors, and are an extension of vectors and matrices to D-dimensional space. These are useful, because for a D-dimensional problem there is one dimension for the different channels of the input data and another for the number of samples. Hence, a D-dimensional problem needs a rank $D + 2$ tensor to store data.

For image data, there are typically three channels; these are the channels corresponding to the red, green and blue values of each pixel, hence the image has to be represented by a rank

three tensor. The data, however, has to be stored in a rank four tensor to account for all the different samples. I.e., a shape of $samples \times i \times j \times 3$, where $i \times j$ is the image size. Similarly, for time series problems, each channel may represent a different variable with a shared time axis. Hence, time series data may be stored in a rank three tensor, $\boldsymbol{X} \in \mathbb{R}^{samples \times L \times N_c}$, where $samples \times L \times N_c$ is the shape; L is the length of each sample; and N_c are the number of channels.

A common characteristic of input data where convolutions are appropriate, is that it has some sort of temporal and/or spatial structure. Adjacent input elements, such as the surrounding pixels in an image, or the previous and next time steps in a time series are usually highly correlated. Hence, using fully-connected layers with exponentially many parameters may be wasted on these types of inputs. This is the obvious advantage of the CNN, where the original input space is reduced to feature maps using many times less parameters.

The cornerstone of the CNN is the convolution operation. A convolution in the ML context involves an element-wise multiplication of input data and a kernel, which are the weights of the convolutional layer. A different kernel is applied to each channel, and the collection of these kernels and the bias is considered to make up *one* filter. The resulting matrix from the convolution is called an output feature map, which similarly to the MLP, is then fed through an activation function before being passed on to the next layer. Moreover, if there are N_f different filters, there are also N_f output feature maps.

A one-dimensional convolution between an $L \times N_c$ input sample and *one* $N_k \times N_c$ filter, where N_k is the kernel size, is illustrated in figure 2.5. The two axes may represent the time and channel axes, moving horizontally and into the screen. The 1D convolution can be visualized as the filter moving along the time axis step by step, and calculating the sum of the element-wise multiplication of the shadowed elements and the corresponding elements of the overlapping filter. The tth element of the jth output feature map can be given by

$$z_j^{(t)} = b_j + \sum_{i=1}^{N_c} \boldsymbol{x}_i^{(t:t+N_k-1)} \boldsymbol{w}_{i,j}^T, \tag{2.35}$$

where $\boldsymbol{w}_{i,j} \in \mathbb{R}^{N_k}$ are the parameters of the ith channel and jth filter; b_j is the bias of the

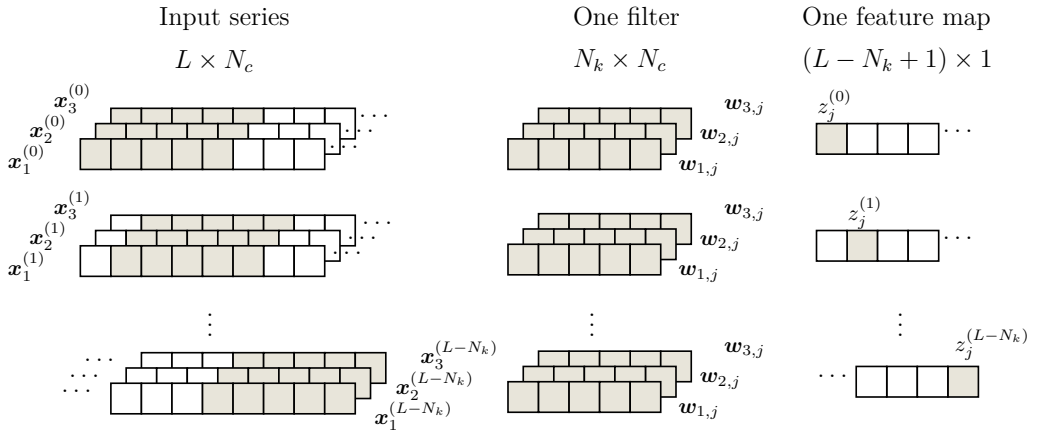


Figure 2.5: Illustration of a one-dimensional convolution operation for an input series and one of the filters. L is the sample length; N_c is the number of channels; and N_k is the kernel size, hence $N_k \times N_c$ is the filter size.

jth filter; and $\boldsymbol{x}_i^{(t:t+N_k-1)}$ is known as the receptive field of the tth element. The receptive field is the slice of values for the ith channel starting at element t, and has the same length as the kernel. The receptive field of each respective feature map element can be seen from figure 2.5 as the shadowed input elements. Notice also how the kernel weights remain the same across all the time steps. This is known as weight sharing and reduces the number of parameters compared to using a fully-connected layer considerably (Gu et al., 2018, p. 2-3).

Note that the convolution shown is with a stride, $s = 1$ and padding, ρ : "valid". These hyper-parameters determine how the filter moves along the time axis, where stride decides the step length of the filter along the time axis for each iteration, while the type of padding determines how zeroed elements are added to each side of the input sample. Valid padding has no zeroed elements, while "same" padding adds zeroed elements to the start and end of the sample. Same padding may be used to ensure that all elements of the input are used. I.e., if $s > 1$ then the last step of the convolution may jump over the extent of the input, meaning that the last elements are never considered (Chollet et al., 2015). Padding may also be used to preserve the spatial dimensions of the input.

The output feature maps of a convolutional layer are usually very large. Pooling layers reduce the size of the input feature maps, determined by the pool size N_p, by applying some statistical function to windows of the feature maps. The pooling procedure can be visualized

in a similar manner as for the convolutional operation shown in figure 2.5. However, instead of using element-wise multiplication, a pooling function is applied to the shadowed elements. Commonly used pooling functions include max pooling and average pooling, where the output is the maximum and average value of the window, respectively. Finally, the addition of a pooling layer to the network is usually followed by another pair of convolutional and pooling layers, or by flattening the final feature maps and feeding the vector to a fully-connected layer (Bishop, 2008, p 269).

2.3.5 Neural network optimization

Neural network optimization can be nuanced due to the size and complexity of modern neural networks. The choice of optimizer algorithm (SGD, mini-batch SGD and variants) and the initialization of parameters is crucial in achieving a stable and converging optimization procedure.

Variants of stochastic gradient descent

Many extensions or variations to SGD have been proposed to improve the stability and learning speed of neural networks (Goodfellow et al., 2016, p. 292). Classical momentum introduces a velocity vector $\boldsymbol{v}$ to the parameter updates (see equation 2.12). There are then two update rules to the algorithm, given by

$$\begin{aligned} \boldsymbol{v}^{\tau+1} &= \beta \boldsymbol{v}^{\tau} - \eta \nabla_{\boldsymbol{\theta}} \ell(\boldsymbol{\theta}^{\tau}; \boldsymbol{x}, \boldsymbol{y}) \\ \boldsymbol{\theta}^{\tau+1} &= \boldsymbol{\theta}^{\tau} + \boldsymbol{v}^{\tau+1}, \end{aligned} \tag{2.36}$$

where β is a hyper-parameter that determines the rate of exponential decay of previous gradients. With momentum, the parameter updates will keep moving somewhat (determined by β) in the direction of previous gradients. This is advantageous when descending error surfaces where the gradient is frequently varying in some directions, while remaining consistent in others. Without momentum, the learning is slowed down by these oscillating gradients and little progress is made (Ruder, 2016, p. 4). With momentum on the other hand, the updates gather momentum in the direction of the most consistent gradients, avoiding the

oscillations.

Nesterov accelerated gradient, or just Nesterov momentum, is an alternative to classical momentum with a slight difference in update rules. They are given by

$$\begin{aligned} \boldsymbol{v}^{\tau+1} &= \beta \boldsymbol{v}^{\tau} - \eta \nabla_{\boldsymbol{\theta}} \ell(\boldsymbol{\theta}^{\tau} + \beta \boldsymbol{v}^{\tau}; \boldsymbol{x}, \boldsymbol{y}) \\ \boldsymbol{\theta}^{\tau+1} &= \boldsymbol{\theta}^{\tau} + \boldsymbol{v}^{\tau+1}, \end{aligned} \tag{2.37}$$

where the only difference between Nesterov and classical momentum, is that the gradient is calculated *after* adding the velocity of the current iteration (Goodfellow et al., 2016, p. 295).

Recall that the learning rate η determines the step size of each iteration. Instead of keeping this hyper-parameter fixed through the entire optimization, it is common to gradually decrease the learning rate as the training progresses. A learning rate schedule, which describes the manner it decreases, is a simple way of changing the learning rate. It is usually chosen to be an exponential or linear decay (Goodfellow et al., 2016, p. 290). A more elaborate approach however, is to use an adaptive learning rate, where the learning rate is adapted to each parameter. I.e., given that the gradient of the cost function may be more sensitive to some parameters than others, it makes sense to decrease the learning rate differently for each parameter (Goodfellow et al., 2016, p. 303). There exist multiple approaches to this method and some frequently used optimization algorithms for neural networks will be briefly described. Similarly to the discussion on activation functions (section 2.3.3), it is important to note that there is no optimizer which is guaranteed to provide the best results for any given problem. There are, however, default recommendations and some optimizers which have been shown to empirically outperform its peers (Goodfellow et al., 2016, p. 306).

Adaptive subgradient descent (AdaGrad) is an algorithm which implements an adaptive learning rate. AdaGrad also uses a global learning rate, but scales it for each parameter based on an accumulation of previous gradients. It is scaled by the inverse square root of accumulated squared gradients, which has the effect of quickly decreasing the learning rate of the parameters with the largest influence on the cost (Goodfellow et al., 2016, p. 303). The downside to the accumulation of squared gradients, is that learning rates may decline to

levels too small to continue learning as training progresses (Ruder, 2016, p. 6). The AdaGrad update rules may be given by

$$\begin{aligned} \boldsymbol{g}^{\tau} &= \nabla_{\boldsymbol{\theta}} \ell(\boldsymbol{\theta}^{\tau}; \boldsymbol{x}, \boldsymbol{y}) \\ \boldsymbol{r}^{\tau} &= \boldsymbol{r}^{\tau-1} + \boldsymbol{g}^{\tau} \odot \boldsymbol{g}^{\tau} \\ \boldsymbol{\theta}^{\tau+1} &= \boldsymbol{\theta}^{\tau} - \frac{\eta}{\sqrt{\boldsymbol{r}^{\tau} + \epsilon}} \odot \boldsymbol{g}^{\tau}, \end{aligned} \tag{2.38}$$

where $\boldsymbol{g}^{\tau}$ is the gradient of the current iteration; $\boldsymbol{r}$ is the accumulated gradients up to the current iteration; and ϵ is a small constant to avoid division by zero.

To alleviate the shrinking learning rate of AdaGrad, both the AdaDelta and root mean square propagation (RMSProp) algorithms implement a *decaying* moving average of the squared gradients instead. RMSProp defines the update rule of the accumulated gradient as

$$\boldsymbol{r}^{\tau} = \rho \boldsymbol{r}^{\tau-1} + (1 - \rho) \boldsymbol{g}^{\tau} \odot \boldsymbol{g}^{\tau}, \tag{2.39}$$

where ρ determines the decay rate (Ruder, 2016, p. 7).

Adaptive moment estimation (Adam) combines some aspects of RMSProp and classical momentum to incorporate both an exponentially decaying average of past gradients (first moment), and of past squared gradients (second moment). Additionally, to avoid the moments tending towards zero when initialized as $\mathbf{0}$, a bias correction is made in the update rules. The rules are given by

$$\begin{aligned} \boldsymbol{s}^{\tau} &= \beta_1 \boldsymbol{s}^{\tau-1} + (1 - \beta_1) \boldsymbol{g}^{\tau} \\ \boldsymbol{r}^{\tau} &= \beta_2 \boldsymbol{r}^{\tau-1} + (1 - \beta_2) \boldsymbol{g}^{\tau} \odot \boldsymbol{g}^{\tau} \\ \hat{\boldsymbol{s}}^{\tau} &= \frac{\boldsymbol{s}}{1 - \beta_1^{\tau}} \\ \hat{\boldsymbol{r}}^{\tau} &= \frac{\boldsymbol{r}}{1 - \beta_2^{\tau}} \\ \boldsymbol{\theta}^{\tau+1} &= \boldsymbol{\theta}^{\tau} - \frac{\eta}{\sqrt{\hat{\boldsymbol{r}}^{\tau}} + \epsilon} \odot \hat{\boldsymbol{s}}^{\tau}, \end{aligned} \tag{2.40}$$

where $\boldsymbol{s}$ and $\boldsymbol{r}$ are the first and second moments; β_1 and β_2 are the decay rates for the first

and second moments, respectively; and $\hat{\boldsymbol{s}}$ and $\hat{\boldsymbol{r}}$ are the bias corrections.

Finally, in practice, Nesterov momentum has been shown to be a better alternative than classical momentum in some cases, which motivated the Nesterov-accelerated adaptive moment estimation (Nadam) algorithm (Ruder, 2016, p. 8). This algorithm is in practice Adam with Nesterov momentum instead of classical momentum.

Parameter initialization

The parameters for the 0th iteration of the optimization have to be specified. The type of parameter initialization may be crucial in order to reach a satisfactory solution. The weights of neural networks (weights in the MLP and the kernel weights in CNNs), are typically chosen randomly from a uniform distribution, while the biases are set to $\mathbf{0}$ or some other heuristically chosen constant (Goodfellow et al., 2016, p. 298). Glorot and Bengio (2010) introduced a popularly used initialization distribution known as the Glorot, or Xavier, uniform distribution. With Glorot uniform initialization, the weights in a layer l are drawn so that

$$w_{ij}^{[l]} \sim U\left[-\sqrt{\frac{6}{n^{[l-1]}+n^{[l]}}}, \sqrt{\frac{6}{n^{[l-1]}+n^{[l]}}}\right], \tag{2.41}$$

where U is a uniform distribution. Note that if $l = L$ then $n^{[l]} = m$ from figure 2.2, and likewise if $l = 1$ then $n^{[l-1]} = n$.

2.3.6 Regularization and dropout

Recall from section 2.2.1 that early-stopping is used to prevent over-fitting to the training data. This is one of many methods for implementing model regularization in the pursuit of reducing over-fitting. Perhaps the most commonly known form of regularization is one that penalizes the norm of the parameters (L^1 and L^2 regularization). This type of regularization adds another term to the cost function (equation 2.10), where large parameter values increase the cost and in turn keep the parameters small (Goodfellow et al., 2016, p. 226).

Another powerful regularization method for neural networks is dropout. For every parameter updating cycle during optimization, units in a layer with dropout have a chance (determined by the dropout rate p_d) to be ignored during that cycle. This means that their activations,

as well as all the connections to other units, are dropped for that particular iteration. As a consequence, by the end of optimization the model may have been trained on many different combinations of units, which prevents the model from relying on a small number of units. This also means that dropout, in essence, is a form of bagging (Goodfellow et al., 2016, p. 255). I.e., it is a type of ensembling where essentially multiple different neural networks are trained and evaluated on the same data in order to reduce bias.

2.4 Explainable artificial intelligence

Explainable artificial intelligence (XAI) has seen an increase in interest the past years (Arrieta et al., 2020). There are differing definitions in the field, as it is still a relatively new area of research. However, a general definition made by (Arrieta et al., 2020, p 6) in a comprehensive taxonomy of the new field is that "given an audience, an explainable Artificial Intelligence is one that produces details or reasons to make its functioning clear or easy to understand". They also categorized the number of different XAI methods into distinct categories based on the type of explanation. These included (1) visualization; (2) model simplification; (3) local explanations; (4) feature relevance; (5) text explanations; and (6) explanations by example. Out of these six, the authors noted that model simplification and feature relevance techniques in particular have received a lot of attention recently, with methods like Shapley Additive explanations (SHAP) and Local Interpretable Model-agnostic Explanations (LIME). These XAI methods also offer open-source Python implementations and onboard visualization which make them accessible to practitioners.

The following sections will investigate post-hoc techniques (meaning XAI methods that are used *after* the prediction model is made), and will particularly focus on feature relevance and model simplification methods that current XAI frameworks offer.

2.4.1 Deep learning important features

Deep Learning Important Features (DeepLIFT), presented by Shrikumar et al. (2019), is a feature relevance method specifically made for neural networks. DeepLIFT assigns contribution scores to the hidden units of any layer (e.g., the input layer) in a neural network by

backpropagating (see chapter 2.3.2) activation differences of its hidden units with respect to some reference. This reference has to be chosen for each particular problem and is important in order to get intuitive explanations. An advantage of DeepLIFT is that it does not use gradients to calculate the contributions of inputs. Notably, in the case of a saturated unit and a zero-gradient, attribution of contributions using the gradient may imply that the input in question had no impact on the output. The difference in reference of some target output unit $\Delta y_t = y_t - y_t^0$ is given by the summation-to-delta property of DeepLIFT:

$$\Delta y_t = \sum_{i=1}^{n} C_{\Delta x_i \Delta y_t}, \tag{2.42}$$

where C is the contribution to Δy_t attributed to Δx_i. Note that equation 2.42 has been slightly modified compared to the original paper to adhere to the notation laid out earlier (refer to section 2.3.1 for MLP notation).

The quotient of the contribution $C_{\Delta x_i \Delta y_t}$ and the difference from reference in some input unit Δx_i is defined as the multiplier given by

$$m_{\Delta x_i \Delta y_t} = \frac{C_{\Delta x_i \Delta y_t}}{\Delta x_i}. \tag{2.43}$$

Analogous to the derivation of the backpropagation rules laid out in appendix A, the chain rule for multipliers in DeepLIFT is given by

$$m_{\Delta x_i \Delta y_t} = \sum_{j=1}^{n^{[1]}} m_{\Delta x_i \Delta a_j^{[1]}} \cdot m_{\Delta a_j^{[1]} \Delta y_t}. \tag{2.44}$$

The authors of DeepLIFT proved that equation 2.44 maintains the summation-to-delta property (equation 2.42) given the definition of multipliers (equation 2.43). The multipliers are given by the definition of rules laid out in the original paper. The multipliers of neighbouring layers (e.g., the input and first hidden layer) are given by the Linear and Rescale rules for

linear and non-linear units, respectively. Multipliers for linear units are given by

$$m_{\Delta x_i \Delta a_j^{[1]}} = \begin{cases} 0.5 w_{ji}^{[1]} & \Delta x_i = 0 \\ w_{ji}^{[1]} & \Delta x_i \neq 0 \end{cases}, \tag{2.45}$$

while multipliers for non-linear units are given by

$$m_{\Delta x_i \Delta a_j^{[1]}} = \begin{cases} \frac{\partial a_j^{[1]}}{\partial x_i} & \Delta x_i \to 0 \\ \frac{\Delta a_j^{[1]}}{\Delta x_i} & \text{otherwise} \end{cases}. \tag{2.46}$$

The authors also introduced the RevealCancel rule to attribute positive and negative contributions differently. However, the Deep SHAP estimation method (covered in section 2.4.3) does not apply this rule. The rules are explained in detail in the original paper (Shrikumar et al., 2019).

With the definition of the linear and non-linear multiplier rules, the contribution of any unit to the change in output can be found by the use of backpropagation and the multiplier chain rule (equation 2.44). The method is as follows: (1) calculate reference activations for all units by a forward pass (equations 2.21-2.23); (2) calculate actual activations and the differences from reference; (3) start with the multipliers of the last hidden layer with respect to the output layer and backpropagate with the use of equation 2.44 to find the multipliers of all units; and (4) find contributions of inputs by applying equation 2.43. This is analogous to the backpropagation rules derived in appendix A.

2.4.2 Shapley values

Shapley values are a concept from cooperative game theory where players (features) work together to form some pay-out (prediction). The Shapley value of the jth feature in the context of a model f and an instance $\boldsymbol{x}$ gives the weighted average of all possible contributions the feature may have to the model output. For all possible coalitions, the prediction of the model is found *with* and *without* the presence of the feature in question. The *exact* Shapley

value is defined by

$$\phi_j = \sum_{S \subseteq F \setminus \{j\}} \frac{|S|!(|F| - |S| - 1)!}{|F|!} [f_{S_j}(\boldsymbol{x}_{S_j}) - f_S(\boldsymbol{x}_S)], \tag{2.47}$$

where S is a subset of the set of features, F without j; $S_j = S \cup \{j\}$ is the subset with j added; f_{S_j} is the model built *with* the jth feature as an input, and f_S is the model built *without* it.

Equation 2.47 implies that the model has to be retrained twice for every subset; for the subset containing j and once without its presence. This is computationally expensive since there will be $2^{|F|}$ different coalitions to evaluate. Some methods have been proposed to reduce the computations required to calculate the exact Shapley value.

2.4.3 Shapley additive explanations

Shapley additive explanations (SHAP) introduced by Lundberg and Lee (2017) are based on so-called SHAP values which are the Shapley values of a conditional expectation function of the original model. SHAP values satisfy three important properties of feature importance that other methods do not:

(1) Local accuracy: the SHAP explanation should match the original prediction

(2) Missingness: missing features in the original input space should have no impact

(3) Consistency: importance attributed to a feature should stay consistent with its contribution to the model

Instead of re-training the model for every subset, SHAP values approximate the model's output for a subset of inputs with the expected value of the original model conditioned on that subset. I.e., define that

$$f_{\boldsymbol{x}}(S) = f(h_{\boldsymbol{x}}(\boldsymbol{z}')) = E[f(\boldsymbol{x})|\boldsymbol{x}_S], \tag{2.48}$$

where $h_{\boldsymbol{x}}$ is the input mapping function and $\boldsymbol{z}' \in \{0,1\}^n$ is a binary vector representing

whether a feature is missing or present (0 or 1); and S is the set of features which are "ON" in $\boldsymbol{z}'$, so that $\boldsymbol{x}_S$ has *missing* values for the features not in S. With this definition, the model does not have to be re-constructed for every subset. Particularly, the output of the model with the absence of some features ($f_S(\boldsymbol{x}_S)$ in equation 2.47), is given by the conditional expectation $E[f(\boldsymbol{x}|\boldsymbol{x}_S)]$ instead.

SHAP defines a missing feature in $\boldsymbol{x}_S$ by extracting a random value from a background set (note the similarity to DeepLIFT's reference values). The background set is meant to represent the expected distribution of the feature so that the missing feature is given some uninformative value. As a consequence, the SHAP value of some feature is interpreted as the contribution to the difference between the actual prediction and the *expected* prediction acquired from the background set (may also be referred to as the baseline prediction). By property 1 a *single* prediction made from an instance $\boldsymbol{x}$ may be given as a sum of SHAP values:

$$f(\boldsymbol{x}) = \phi_0 + \sum_{i=1}^{n} \phi_i, \tag{2.49}$$

where ϕ_0 is the baseline prediction $\phi_0 = E[f(\boldsymbol{x}')]$, interpreted as all the features missing from the input space ($\boldsymbol{x}' = h_{\boldsymbol{x}}(\mathbf{0})$); and ϕ_i is the SHAP value of the ith feature. By explaining the prediction as a sum of SHAP values, each SHAP value gives an idea of the feature's impact on the particular instance.

The conditional expectation function (equation 2.48) is still computationally intensive to calculate exactly. Hence, the authors of the original paper also laid out several different ways to estimate the SHAP values. The methods may be distinguished by whether they are model-specific or agnostic. Of special interest is the Deep SHAP estimation method, specific to neural networks, which unifies DeepLIFT's feature attribution with SHAP values. In particular, let $x_i^0 = E[x_i]$ and $y_t^0 = E[y_t]$; i.e., that the references are given by the expected values provided by the background data. Furthermore, interpret the DeepLIFT contribution as the SHAP value of the ith input on the tth output unit, then $C_{\Delta x_i \Delta y_t} = \phi_{i,t}$. Lastly, assuming feature independence and model linearity in equation 2.48 gives $E[f(\boldsymbol{x})|\boldsymbol{x}_S] \approx f(\boldsymbol{x}_{\boldsymbol{S}}, E[\boldsymbol{x}_{\bar{S}}])$. With these definitions, a slightly modified multiplier from equation 2.43 was given by Lundberg and Lee (2017) in terms of the SHAP value and difference from the

expected value of the background data:

$$m_{x_i y_t} = \frac{\phi_{i,t}}{x_i - E[x_i]}. \tag{2.50}$$

The Deep SHAP values may then be calculated similar to the calculation of DeepLIFT contributions described in section 2.4.1 through backpropagation of multipliers, and the linear and non-linear rules. Since Deep SHAP assumes feature independence and lienarity, Deep SHAP is a model simplification and feature attribution method. As a consequence of the feature independence assumption, the feature attribution of correlated features have to be scrutinized.

While the original SHAP paper demonstrated (by linearizing the Max function) how analytical solutions of linearization rules for each activation function can be produced, the open-source implementation of Deep SHAP (Lundberg and Lee, 2017) appears to be using the DeepLIFT Linear and Rescale rules for linear and non-linear activation functions. In practice, the main difference between DeepLIFT and Deep SHAP therefore seems to be the choice of background data. Particularly, the fact that the original DeepLIFT method only uses *one* reference sample, while Deep SHAP relies on the background data providing a distribution of "typical" data in which SHAP values can be estimated from. DeepLIFT attributes feature importance based only on the one reference, while Deep SHAP computes SHAP values by averaging attributions over all the background samples. Hence, Deep SHAP is essentially DeepLIFT when only one background sample is provided as background data.

Chapter 3

Data collection and analysis

It is often said that the results of a prediction model is only as good as the quality of the provided data. Hence, time spent extracting, analyzing and cleaning relevant data is not time wasted. In this chapter the data sets used in the thesis are described and analyzed. Particularly, the methods for acquiring the data will be described; the ways in which missing or invalid data was replaced is shown; and the data itself is visualized and analyzed in detail.

3.1 Norwegian electricity consumption data

Norway is split into five different price areas (NO1, NO2, NO3, NO4 and NO5), and the chosen price region for the case study in the thesis was NO1. This is shown in figure 3.1. According to the Norwegian TSO, Statnett (2021a), the hourly aggregated electricity consumption of one price area is calculated as the sum of production and energy flow (in or out) of a particular price region and hour (by energy balance this has to match the consumption). Then, theoretically, the aggregated consumption of one price region may be given by

$$E^{(t)} = \int_t^{t+3600} P(t)dt = \int_t^{t+3600} (G(t) + \sum I_{ij}(t))dt, \tag{3.1}$$

where $E^{(t)}$ denotes the discrete aggregated consumption of some hour t in watt-seconds; $P(t)$ denotes the instantaneous active power load of some price region as a function of time; $G(t)$ the instantaneous active power generation; and $\sum I_{ij}(t)$ is the sum of instantaneous

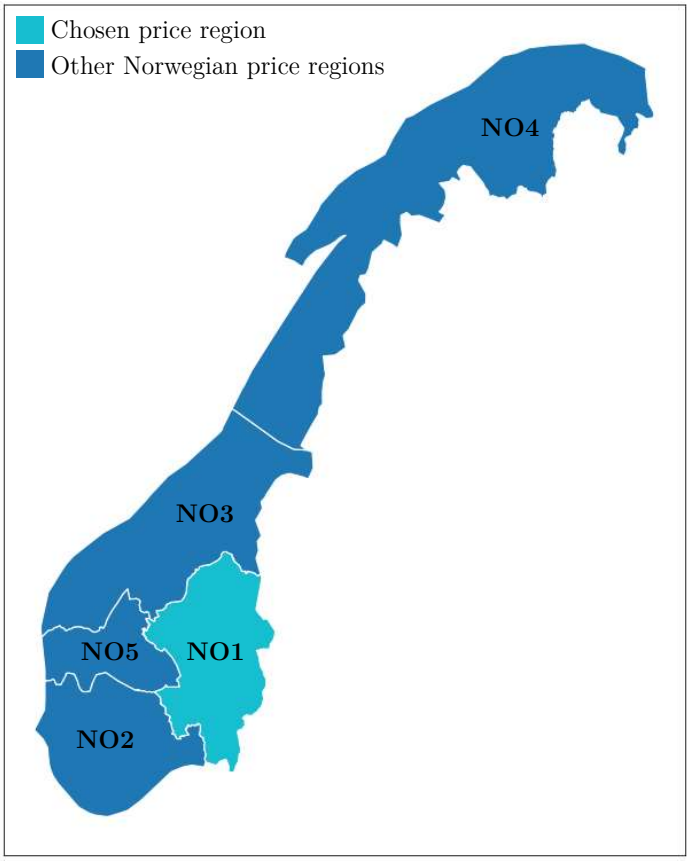


Figure 3.1: The Norwegian price areas and the selected price area. Figure adapted from original work by Ugur Halden.

power flows in and out of the price region. Note that the power flows are not limited to the Norwegian price areas, but includes power flow from HVDC interconnectors and power flow to Swedish price regions.

Let the hourly average of the instantaneous active power load of some hour t be denoted by $\bar{P}^{(t)}$, or $P^{(t)}$ for convenience, then

$$P^{(t)} = \frac{1}{t + 3600 - t} \int_{t}^{t+3600} P(t)dt = \frac{1}{3600} E^{(t)}. \tag{3.2}$$

If $E^{(t)}$ is given in watt-hours, then equation 3.2 reduces to $P^{(t)} = E^{(t)}$. I.e., the hourly average active power load, or just power load, of some price region is given by the hourly aggregated consumption.

Table 3.1: The structure of historical consumption data in megawatt-hours (MWh).

	Hours	NO1	NO2	NO2	NO3	NO4	NO5	NO
01/01/2014	00 - 01	3917	3843	2424	2158	2032	14374	14374
01/01/2014	01 - 02	3862	3817	2342	2151	2009	14181	14181
01/01/2014	02 - 03	3767	3756	2284	2149	1981	13936	13936

3.1.1 Data extraction and description

Hourly aggregated consumption data for the Norwegian price areas from as far back as 2013 is publicly available from Nord Pool (2021). A few samples of the data structure from 2014 is shown in table 3.1. Each row in the data set represents the aggregated electricity consumption of the coming hour in local time. Since the consumption data is given in megawatt-hours, the numerical value is also a measure of the hourly average power load for that hour given in megawatts, as shown in equation 3.2. This quantity will henceforth be referred to as the electrical load, power load or just load for the sake of simplicity. However, it is important to remember that it actually refers to the hourly *average* active power load. It does not give any information regarding instantaneous values such as the maximum (peak) or minimum power load of the hour.

3.1.2 Seasonality and statistics

Data from 2014-2020 in NO1 was extracted as the power load data. Descriptive statistics of this data set is shown in table 3.2. As seen from the total count, there are no missing data for this data set. The mean has seen an increase after 2015, but fell back to previous levels in 2020. The standard deviation also varies from year to year, but does not appear to be increasing significantly. Moreover, the percentiles (25%, 50% and 75%), minimum and maximum statistics also show no obvious changes. These observations may indicate that the load has not seen any increase or decrease over the years.

This is substantiated from the yearly time series in figure 3.2. I.e., that there is no obvious increasing or decreasing trend for the power load of NO1. Apart from 2020, which appears to have unusually low winter load and also slightly lower load in general, all other years have

Table 3.2: Descriptive statistics of the NO1 consumption data.

Year	Count	Mean [MW]	Std [MW]	Min [MW]	Max [MW]	25% [MW]	50% [MW]	75% [MW]
2014	8760	3932	1269	1678	7478	2891	3756	4889
2015	8760	3997	1181	1870	7414	3064	3837	4858
2016	8784	4156	1403	1755	8133	3001	3981	5182
2017	8760	4165	1319	1802	7615	3072	3996	5193
2018	8760	4177	1472	1699	7712	2916	3934	5450
2019	8760	4117	1298	1693	7624	3035	3988	5142
2020	8784	3952	1100	1812	6846	3052	3852	4832
All	61368	4071	1301	1678	8133	3009	3893	5067

similar peaks and troughs. This may seem counter-intuitive since electricity consumption in Norway is expected to continue growing (Spilde et al., 2019). DSOs in the NO1 region however, note that the local growth appears to be stagnating due to increased penetration of district heating, greater energy efficiency and better thermal insulation in dwellings (Elvia AS, 2020, p. 15). Hence, the consumption patterns of NO1 do not seem to have changed much from 2014 to 2020.

Power load does however contain several distinct seasonalities (see section 2.1). These are the yearly, weekly and daily cycles, and can be seen from figure 3.2. The power load time series of 2016 in figure 3.2 has been split into the four seasons observed in Norway. There are no official definitions of when one season ends and another begins. They may also vary from one year to another and have no clear beginnings or ends as indicated by using dashed lines. Hong (2010) discussed the use of more than the four traditional seasons by also including transitional periods between each season. The four seasons in figure 3.2 were defined as

(1) Winter: December, January and February

(2) Spring: March, April and May

(3) Summer: June, July and August

(4) Autumn: September, October and November

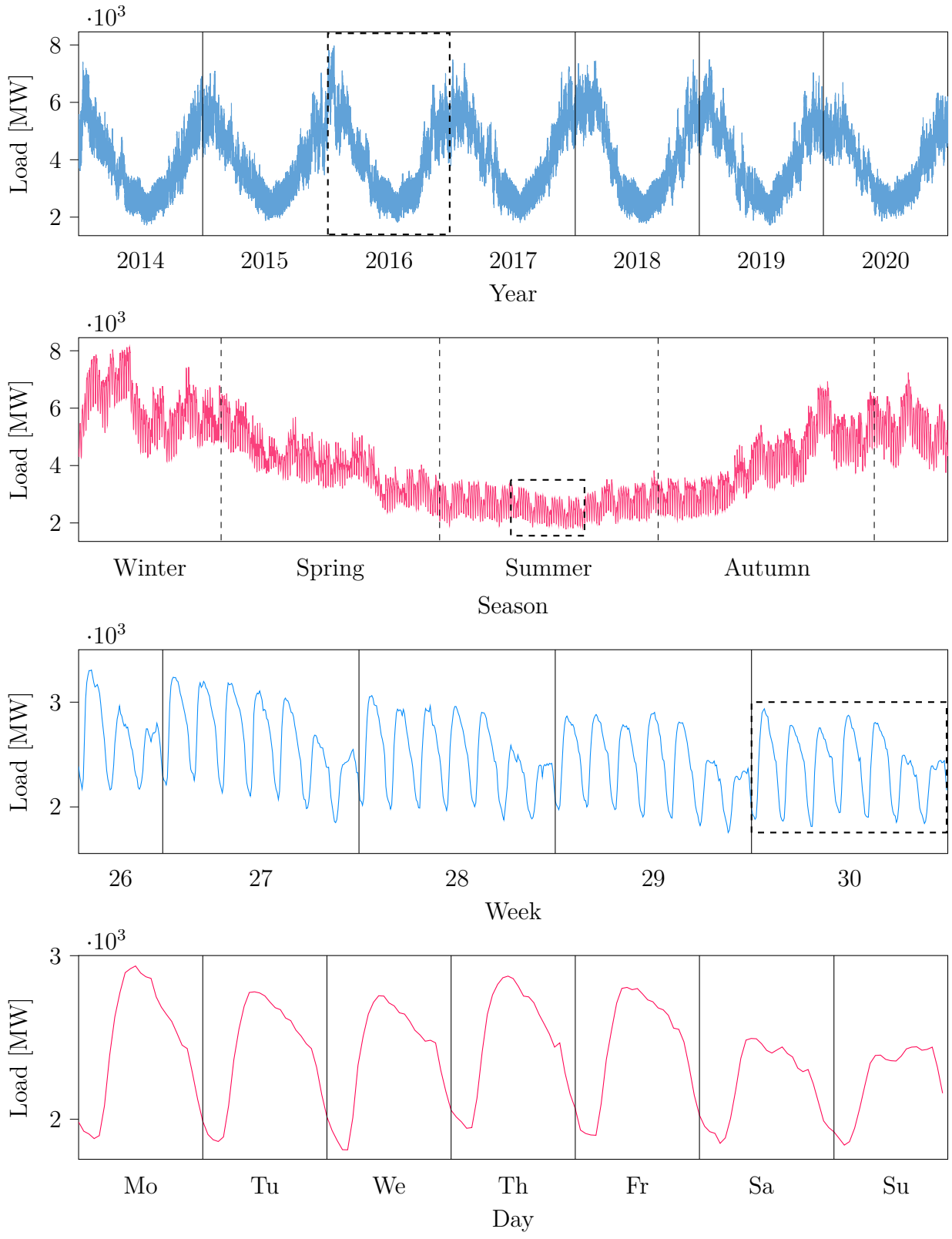


Figure 3.2: Visualization of the electrical load data at different time scopes. The different time scopes show that electrical load is similar from year to year; electrical load has a yearly cycle, a weekly cycle and a daily cycle.

This definition was based on observations of the yearly power load cycle and the meteorological seasons.

During winter in Norway, sunlight is minimal and temperatures are low, which causes higher demand from the use of electrical heating appliances. An unusually mild winter (as was observed in 2020) is therefore also observed as lower power load than previous years. The transition from winter to spring is marked by increasing temperatures and longer days, hence power load gradually decreases until summer. During summer, temperatures are at their highest, hence power load is not as sensitive to temperature fluctuation. This is in contrast to some power systems closer to the equator where heavy use of air conditioning during summer causes the power load to be at its highest (Hong, 2010, p. 44). The load is usually at its lowest in July, particularly during the general staff holidays (*fellesferie* in Norwegian) when large parts of the population go out of country, to their cabins, etc. When summer turns to autumn the days grow shorter and temperatures gradually diminish until winter has arrived. This is yet again observed as increasing electrical load until winter time.

It is well known that electrical power load has a distinct weekly seasonality (Hong, 2010). Each day of the week have slightly different profiles, but the most obvious differences are seen when comparing workdays to the weekend. The weekly cycle can be observed from July 2016 in figure 3.2. Mondays usually have the highest peak power loads and are followed by the rest of the weekdays. When the weekend arrives, the power load drops substantially and remains this way until the next Monday. During the weekend, the power load curve has lower peaks and slightly lags the weekday curves. I.e., the peaks arrive later in the day and the power load remains high for a longer time in the evening. The daily cycle of power load may be seen from week 30 in figure 3.2. During the night, power load is at its lowest around 3AM and rapidly increases as people wake up and the work day begins. The load reaches a peak around 8AM, gradually decreases throughout the day, but remains relatively high. Late in the afternoon there is another peak as people arrive home. Finally, during the evening, the power load has a sharp drop as the day ends and people go to sleep. Figure 3.3 shows three samples of load curves for each day of the week, extracted from three consecutive weeks in July 2016. Notice how the curves are very similar from week to week. Without

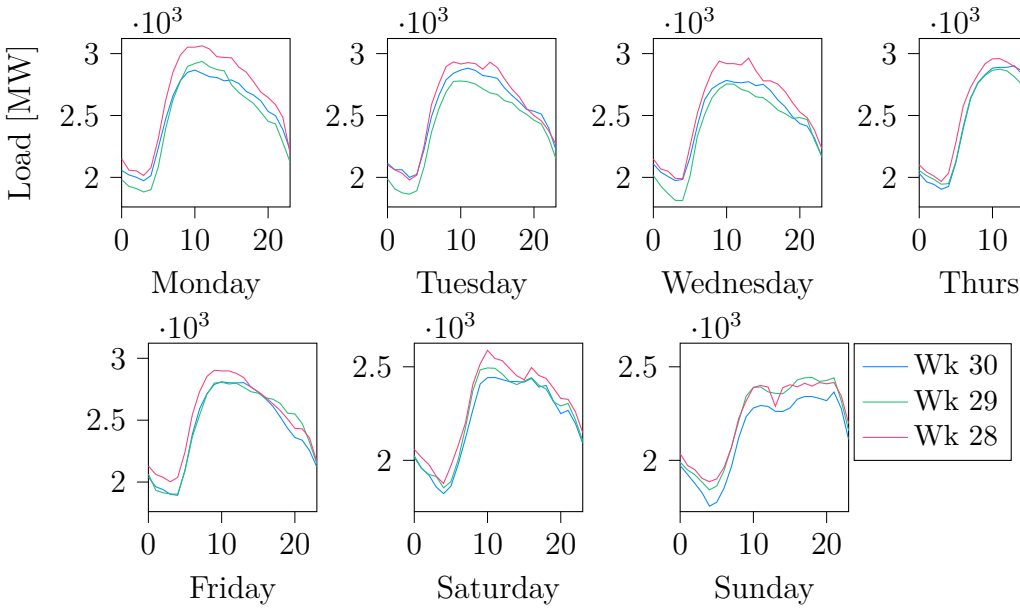


Figure 3.3: The load profiles of the seven days of the week. The individual load profiles are taken from three consecutive weeks in the summer of 2016.

much contribution from the sensitivity to weather variables (since it is summer time), the power load remains almost the same from week to week. This is not the case in every season however, as the load profiles also vary with the time of the year.

The different seasonal load profiles for each day of the week can be seen more clearly from figure 3.4. It shows the average load profiles of the days of the week for the four seasons in 2016. During winter there are two distinct peaks in the power load; one in the morning and another in the afternoon. Conversely, spring tends to have only one peak, with the load profile flattening towards the afternoon. During summer the average load profile has a smoother transition from morning to evening than winter and spring. Lastly, in autumn there appears to be a transition towards two daily peaks again. Interestingly, Sundays do not appear to follow this pattern and has a more consistent load profile across the year. Three important general observations can be made from this: (1) when averaged across each season, the weekdays have almost identical load profiles; (2) weekend load profiles lag the weekday curves by some hours and have lower peaks; and (3) Saturday and Sunday have similar, but slightly different load profiles

As described in section 2.1.1, a PACF plot of a time dependent variable shows the correlation

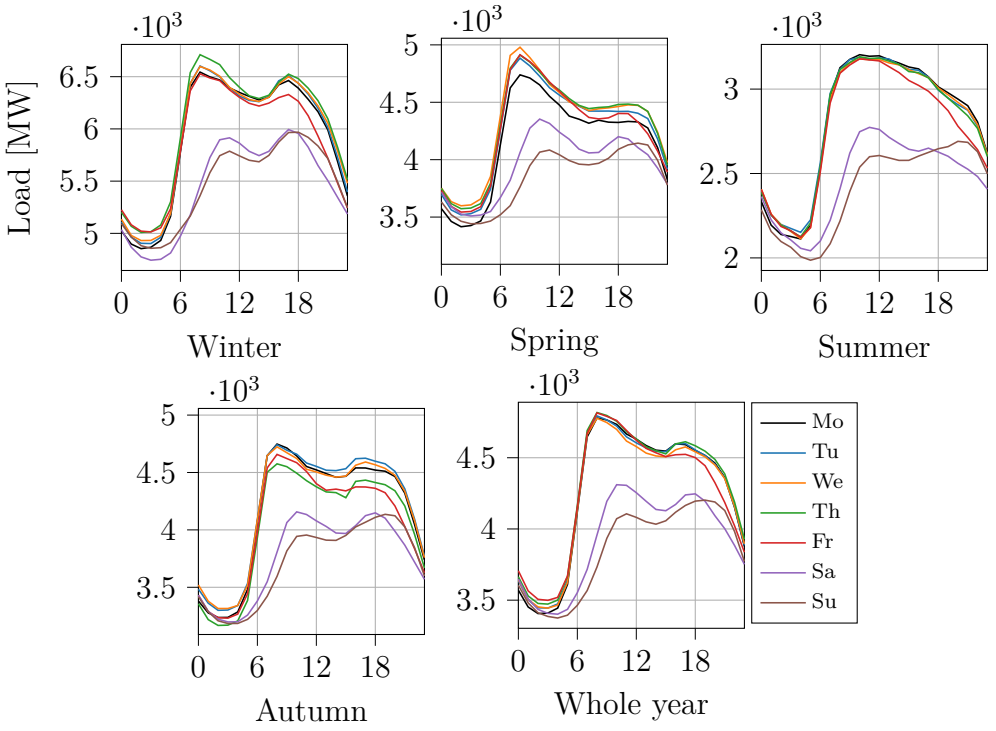


Figure 3.4: Average load profiles of the seven days of the week by season in 2016.

to lagged versions of itself in isolation (i.e., after removing the effects of correlation to other lags). The PACF plot of the power load data set is shown in figure 3.5 up to two weeks back ($k \in [0, 337]$). Recall that he forecasting horizon is day-ahead, hence the first 24 lags of the *target* will never be available at termin time (the boundary is shown by the dashed line). This is unfortunate since lags close to the target hour show almost perfect values of r_k (1 and -1). Notice however, that the largest correlations are found at and around multiples of 24. Particularly, six ($k = 144$) and seven ($k = 168$) days back. Notice also, that although the correlations around two weeks lag ($k = 336$) are significant, most of the correlated lags appear in the range of $k \in [0, 220]$. These findings are consistent with the observations made from visual inspection of the power load curves. I.e., the fact that electrical load follows the daily, and weekly cycles.

3.1.3 Holiday effect

The load profiles discussed above may change entirely in the light of public holidays or other events which change consumption patterns. Table 3.3 shows the most important holidays in

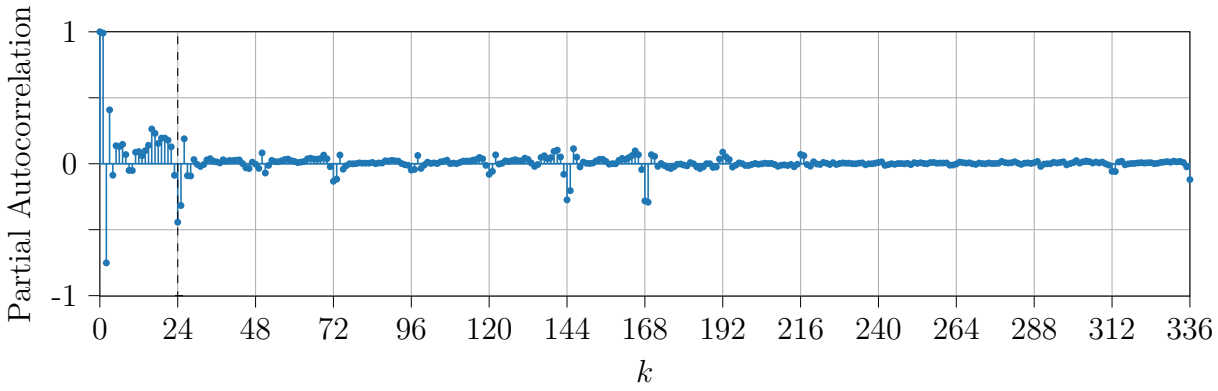


Figure 3.5: Partial autocorrelation plot of the historical load data. Due to the forecasting horizon, only the lags to the right of the dashed line will be available at termin time.

Norway. They may be distinguished by whether or not they are official (public) or observed holidays. Observed holidays are not law-given days off, but are celebrated nonetheless and may possibly have an impact on the elecrical load (e.g., New Year's Eve and some days during Easter). As seen from the table, the months of March, April, May and December in particular have quite a few public holidays in Norway.

Table 3.3: Public holidays and *observed* holidays in Norway

Holiday	Date(s)	Comment(s)
New Year's Day	January 1st	None
Easter Holidays	Late March-early April	Holidays: Maundy Thursday, Good Friday, Easter Sunday, Easter Monday Observed: Palm Sunday, Easter Eve
Labour Day	May 1st	None
Ascension Day	39 days after Easter	None
Constitution Day	May 17th	National day of Norway.
Pentecost Sunday	49 days after Easter	None
Whit Monday	50 days after Easter	
Christmas Eve	December 24th	Restrictions after 3PM.
Christmas Day	December 25th	None
Boxing Day	December 26th	None
New Year's Eve	December 31st	Restrictions after 6PM.

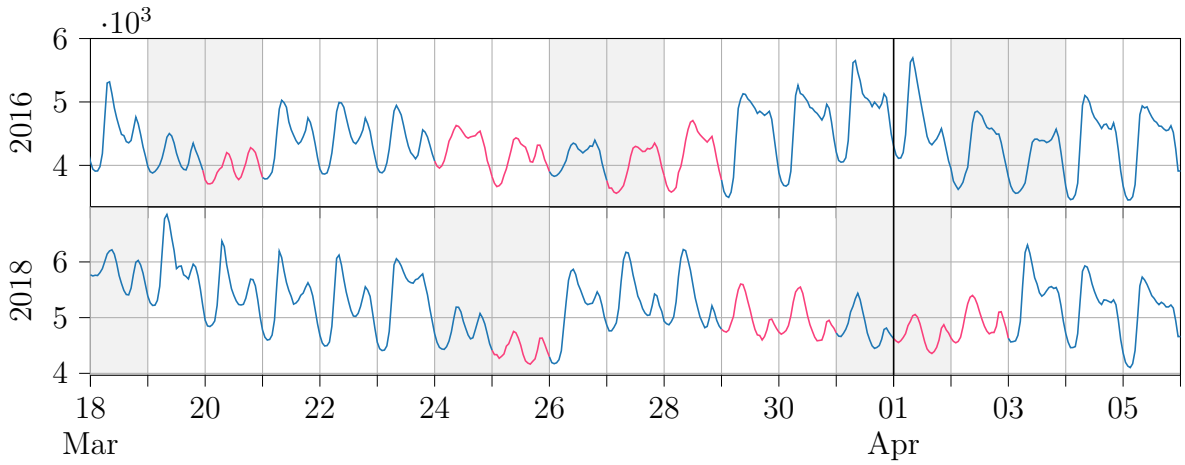


Figure 3.6: Power load in March-April and holidays 2016 and 2018. These months show the influence of Easter on power load.

In order to understand how the different holidays affect electrical load, they have been outlined in figures 3.6-3.8. The three figures show the observed power load for March-April, May, December-January 2016 and 2018 respectively. Holidays are marked in red, while regular days are in blue. Additionally, weekends are highlighted with a gray-shaded background. Although the impact of each holiday may differ from year to year, some general observations about the changes in load profiles can be made from these two years.

Easter took place late March in 2016 and early April in 2018. These two months are shown in figure 3.6. Palm Sunday (20th/25th), Maundy Thursday (24th/29th), Good Friday (25th/30th), Easter Sunday (27th/1st) and Easter Monday (28th/2nd) have been outlined. Initially, it appears that the entire weekend before Easter has very low load in both years. Moreover, Easter in general appears to have lower load and some changes in power load profiles from Palm Sunday through Easter Monday. In particular the peaks are less pronounced and the Easter load profiles observed in 2018 look like none of the load profiles pictured in figure 3.4. Although the official holidays do not include Palm Sunday and Easter Eve, the power load appears to be affected these days as well.

May 2016 and 2018 are shown in figure 3.7. The outlined holidays are Labour Day (1st), Ascension Day (5th/8th), Pentecost (15-16th/20th-21st) and the Constitution Day (17th). In 2016 Ascension Day, Whit Monday and Constitution Day were on a Thursday, Monday

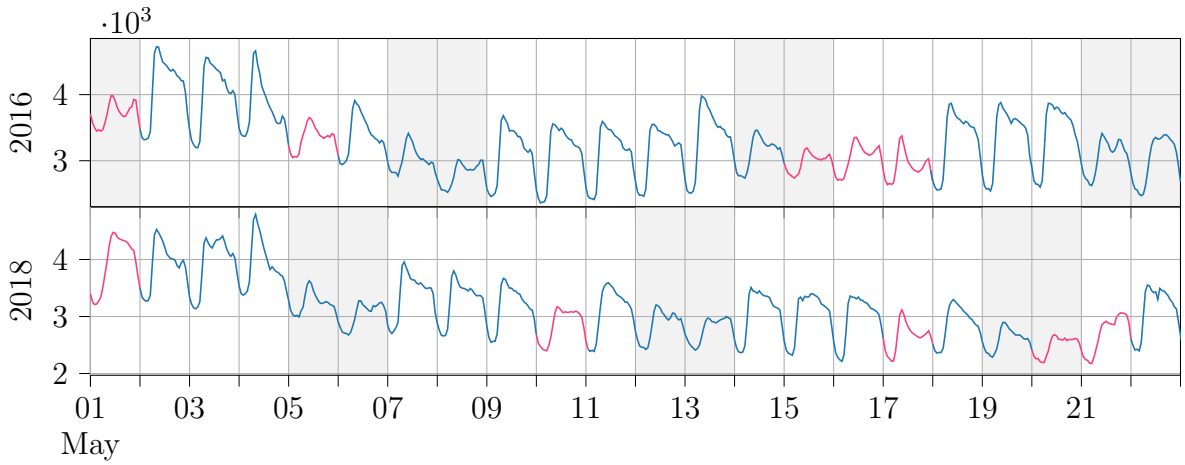


Figure 3.7: Power load and holidays in May 2016 and 2018. May is a month with many holidays in Norway.

and Tuesday, hence it can be seen that these days had reduced power load compared to a normal weekday. Moreover, Friday 6th appears to have been affected by the preceding day being a holiday. In fact, this was what is known as a "squeezed" day in Norway, where many may have chosen to take the day off work for an extra long weekend. This effect is not as easily seen in 2018 however, which may mean the observation made for 2016 is due to something else. May 18th 2018 was also a squeezed day, and may have been affected by this as well. Meanwhile, Labour Day and Pentecost Sunday were both on Sundays in 2016, and therefore did not seem to affect the power load profile much. It is also seen that Labour Day in 2018, even when on a Tuesday, did not affect power load by much. Finally, Constitution Day stands out from the other holidays with sharp peaks at the start of the day and quick drops in the afternoon in both years.

Christmas holidays from 2016 and 2018 are outlined in figure 3.8. Christmas Eve and New Year's Eve are not public holidays, but are included regardless. In 2016 they both fell on Saturdays, and for both days it appears that the evening power load was increased and had sharper peaks than usual. The same can be observed in 2018. From knowledge of what people are doing on these days, this makes sense. Moreover, the curves for New Year's Eve and New Year's Day appear to be slightly lagged compared to the usual pattern. This might be attributed to the fact that people stay up late for New Year's Eve, and get up much later the day after. Another observation which can be made about the Christmas holidays, is

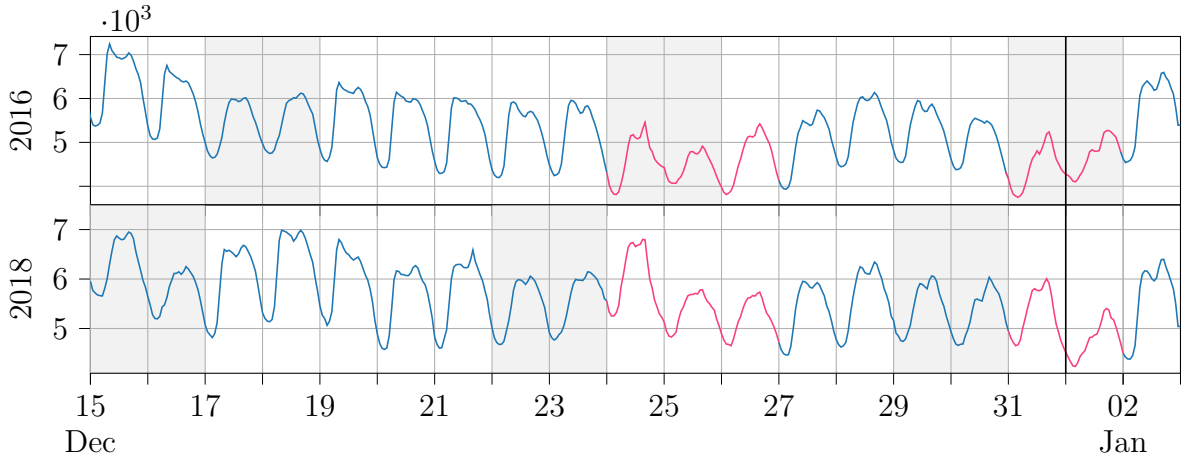


Figure 3.8: Power load and holidays in December-January 2016 and 2018.

that the load on the days in between Boxing Day and New Year's Eve appear to be slightly affected by the holiday season, but not by much. This is given by the fact that most people are back at work these days, but not everyone.

In general, some observations about the impact of holidays on the power load in 2016 and 2018 can be made:

(1) if a holiday falls on a weekday, the power load is generally lower

(2) if a holiday is in the weekend, the load profile does not necessarily change

(3) some holiday load profiles that fall on weekdays resemble the weekend load profiles

(4) other holidays have unique load profiles that only appear on one day.

3.1.4 Daylight saving time

In Norway, the practice of daylight saving time (DST) is carried out on the last Sunday in March and the last Sunday in October. The practice involves turning the clock forward by one hour in March (UTC+1→UTC+2) and backwards by one hour in October (UTC+2→UTC+1), thereby effectively shortening the day in March and extending it in October. Since the power load data follows local time, for every year there is one day in March with 23 recorded values and one day in October with 25 recorded values. The effect of DST on the electrical load data set is shown in table 3.4.

Table 3.4: Daylight saving time in the electrical load data set.

	Hours	NO1	NO2	NO3	NO4	NO5	NO
30.03.2014	01 - 02	3460	3584	2324	2067	1831	13266
30.03.2014	02 - 03						
30.03.2014	03 - 04	3420	3541	2320	2014	1766	13060
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
26.10.2014	01 - 02	2986	3221	2218	1918	1801	12144
26.10.2014	02 - 03	2875	3216	2118	1708	1730	11647
26.10.2014	02 - 03	2831	3183	2063	1695	1737	11508
26.10.2014	03 - 04	2849	3147	2225	1913	1788	11922

3.2 Numerical weather prediction

The following sections cover how the numerical weather predictions were extracted, which variables were selected and a description of them, and the area they cover. A preliminary discussion and analysis of the dependence between the NWP variables and the electrical load is also presented.

3.2.1 Data extraction and description

Historical NWP data from 2012-d.d. is publicly available from the Norwegian Meteorological Institute's (MET Norway) THREDDS data server[1] (TDS). Data is stored in NetCDF format and may be extracted using the OPeNDAP protocol. The current (2016-d.d.) weather forecasting model for the Nordic area (figure 3.9) is the MetCoOp Ensemble Prediction System (MEPS). Forecasts are made up to 60 hours from the termin time depending on the ensemble member with an hourly time resolution. It has a horizontal spatial resolution of 2.5 kilometers, 65 vertical levels and offers a wide range of meteorological variables. Forecasts are ran each day at 00:00, 06:00, 12:00 and 18:00 UTC and have a runtime of about 2 hours and 30 minutes (Frogner et al., 2019). Prior to 2016, MET Norway's forecasting model was the AROME MetCoOp (2014-2016).

[1] https://thredds.met.no/thredds/metno

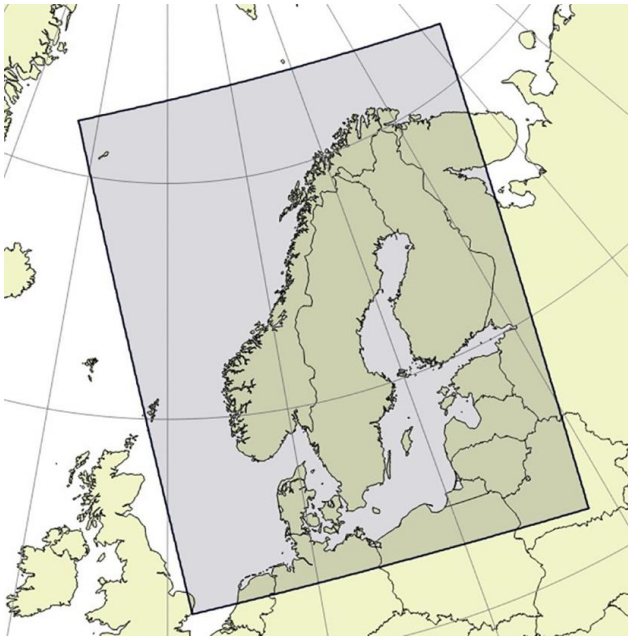


Figure 3.9: The domain covered by the MetCoOp Ensemble Prediction System. Figure from Frogner et al. (2019).

Hourly historical NWP data from the TDS was collected for ten different locations in the NO1 price area from 2014-2020. The locations, and their respective subscripts and coordinates are listed in table 3.5. The locations were mainly chosen based on the dwelling and holiday houses density from Statistics Norway[2] (SSB). However, given the fact that this would have ruled out any cities in Innlandet, some cities representative of the northern parts of NO1 were also selected. Since it is well known that power load responds to the temperature, the hypothesis was that the weather in the most densely populated areas would influence the load the most. Industrial loads, in contrast to residential loads, are considered to be uninfluenced by temperature (Hong et al., 2015), hence they were not considered when choosing locations.

Apart from the impact of temperature, works in the literature have noted the influence of several other weather variables and combinations of them (Apadula et al., 2012). Therefore, some other meteorological variables were also extracted from each location. The selected NWP variables and a description of them is given in table 3.6. Note that while each variable is the average of some hour, it was unclear whether it is the average of the *past* or *coming*

[2]https://kart.ssb.no/

Table 3.5: The selected numerical weather prediction locations.

Location	Subscript	Coordinates (lat, lon)
Drammen	D	(59.747, 10.182)
Fagernes	F	(60.989, 9.267)
Halden	Hl	(59.132, 10.385)
Hamar	Hm	(60.824, 11.082)
Hønefoss	Hø	(60.170,10.259)
Kongsberg	Kb	(59.663, 9.641)
Kongsvinger	Kv	(60.283, 11.987)
Lillehammer	L	(61.115, 10.467)
Moss	M	(59.442, 10.634)
Oslo	O	(59.923, 10.865)

Table 3.6: A description of the selected numerical weather prediction variables.

Name	Symbol	Unit	Description
Air temperature	T	K	Average air temperature 2m above ground level
Relative humidity	H	%	Average relative humidity 2m above ground level
Air pressure	p	Pa	Average surface air pressure
Wind speed	v	m/s	Magnitude of average wind vector ($\|[v_x, v_y]\|$)
Cloud Cover	c	%	Total cloud cover for all heights

hour. However, given that there is some sort of time delay between changes in the weather, and seeing the effect this has on the power load, this was not deemed to be a problem. I.e., with changes in weather variables, particularly temperature, it may take some time (e.g., due to thermal inertia) that changes in power load are observed. Note that the wind speed predictions used in the thesis refer to the magnitude of the wind speed vector $v = |\boldsymbol{v}| = ||[v_x, v_y]||$. Other notable weather variables, like solar irradiation and precipitation, were unfortunately unavailable in some of the chosen NWPs and were therefore not included.

The NWPs were extracted in a manner akin to how they would have been in deployment. I.e., at the termin time of the power load forecast (00:00 Norwegian local time), the latest

Table 3.7: Sample of the data structure of the numerical weather prediction data. The samples are taken from the Kongsberg data.

Time	Temp	Hum	vy	vx	Press	Cloud
03/01/2016 00:00	266.4594	0.882161	0.285661	-3.46811	99758.75	1
03/01/2016 01:00	266.4266	0.927784	-1.30226	-3.44592	99700.69	1
03/01/2016 02:00	266.3753	0.912602	-0.94434	-3.9501	99636.13	1

available forecast from the TDS is the one made at 18:00 UTC. Hence, these historical NWPs were extracted for all dates ranging back to 2014. This choice meant that the NWP data were essentially forecasts with an horizon of 30 to 54 hours. If the 18:00 forecast was not available, the previous forecast (12:00) was chosen instead. The raw data structure for the extracted Kongsberg NWPs is shown in table 3.7. The other locations have the same data structure.

Across all years there were a total of $248 + 72 + 30 + 48 + 0 + 72 + 0 = 470$ missing or invalid (e.g., $T = 9.97 \cdot 10^{36}$) timestamps for all the variables. The year of 2014 had the largest amount of missing data, which was attributed to the last four hours of every day in January missing. It is unknown why this occurred, but it may have been due to a change in forecast horizon of the MET forecast system. These add up to less than 1% of all the data and were therefore imputed using the **interpolate** method from the pandas Python library (McKinney, 2010). Figure 3.10 shows the forecast temperature in Oslo for January 2014 with actual forecast values and imputed values. For short missing segments with steep slopes, the imputed values fit fairy well. The missing horizontal segments however look more conspicuous. Regardless, these were left as they are due to the relatively small amount of missing data. Note that all the other meteorological variables had the same missing data and were imputed in the same fashion.

3.2.2 Dependency between the variables

To get an idea of the amount of information held by the selected meteorological variables, dependency between the variables was measured by calculating MI (see section 2.1.2). Table 3.8 shows the resulting MI-matrix by using the non-parametric estimation (the entropy

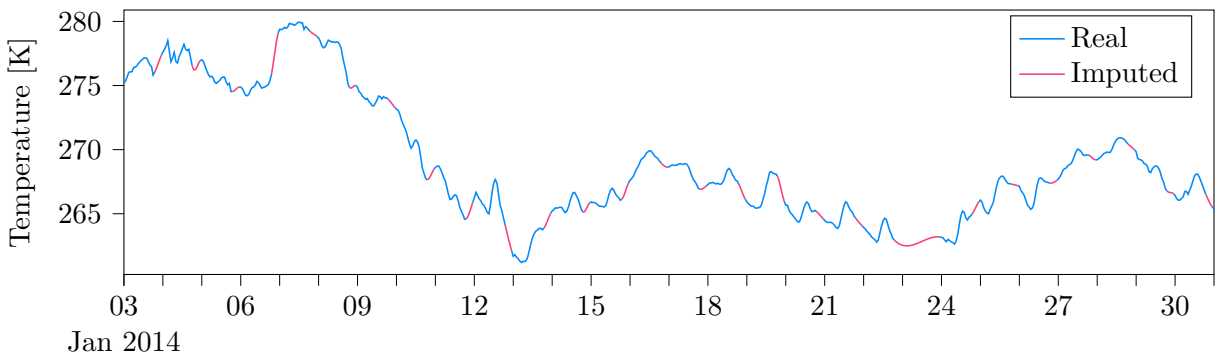


Figure 3.10: Imputation of the numerical weather prediction data

Table 3.8: Mutual information between all of the continuous variables in Oslo.

y \ x	P	T_O	H_O	v_O	p_O	c_O
P	8.46					
T_O	0.70	9.58				
H_O	0.13	0.27	9.59			
v_O	0.02	0.03	0.05	9.60		
p_O	0.11	0.14	0.06	0.05	9.48	
c_O	0.02	0.04	0.16	0.02	0.04	5.24

from equation 2.2 has to be estimated) method **mutual_info_regression** from scikit-learn (Pedregosa et al., 2011). MI coefficients separated by each city were similar for every variable, hence the meteorological time series in the table are the ones from a single city (Oslo). Obviously, the MI coefficients contained in the variables themselves (e.g., $I(P;P)$) are relatively high, and may serve as a reference for the other coefficients. As expected, $I(P;T)$ has the highest coefficient of the meteorological variables. The values of $I(P;H)$ and $I(P;p)$ also suggest that humidity and pressure may contain information about the power load. Note however that the values of $I(H;T)$ and $I(p;T)$ indicate some relationship to temperature. I.e., the temperature by itself may contain some of the same information that humidity and pressure do. This is intuitive given the close relationship between the three variables. The other variables, wind speed and cloud cover, sadly do not seem very promising in terms of MI coefficients.

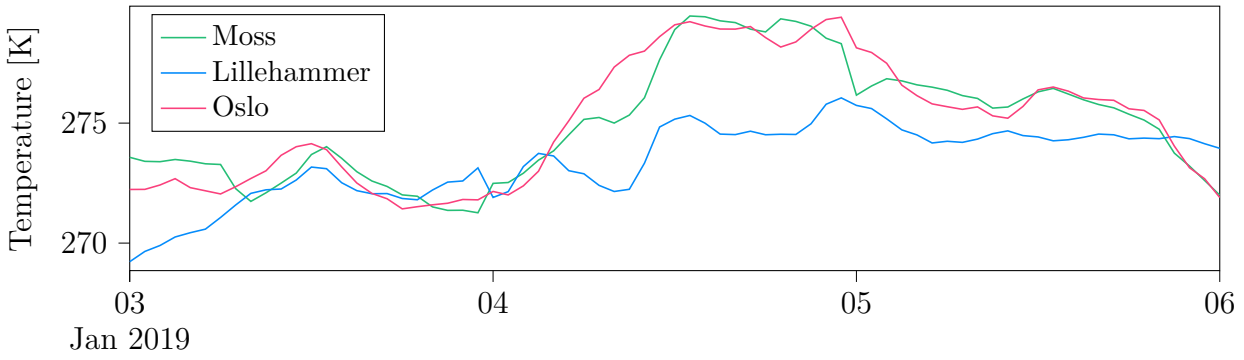


Figure 3.11: The temperature curves of three different locations

Since some of the locations are very close to each other there is a lot of mutual information between them. Figure 3.11 shows the simultaneous temperature predictions in three different locations. As can be seen, the temperature predictions for Oslo (T_O) and Moss (T_M) are very alike, which is likely due to their close proximity. Meanwhile, the predictions for Lillehammer, which is situated further north, are similar, but differ by a few degrees most days. Likewise, the other weather variables see such differences for each location due to the particular local climate and altitude. The main takeaway however, is that each location share a lot of the same information, hence careful consideration has to be made during the feature engineering part of the load forecasting model so as to avoid superfluous features.

Chapter 4

Methodology

This chapter describes the experimental setup; the forecasting problem and strategy; all the developed models; testing of the final model; and how XAI was used on forecasts made by the models.

4.1 Experimental setup

For large parts of the semester, code development and the various experiments were carried out on an ASUS UX303UB laptop from 2016. The CPU was an Intel Core i7-6500U with a core speed of 2.5 GHz. As the amount of processor intensive activities (larger models, more data, hyper-parameter searches, test methods, etc.) grew, there was a need for more processing power to speed up this part of the work. Eventually access was granted to NTNU's Tesla farm, which consists of four Tesla V100-SXM2-32GB GPUs in parallel. In particular these were used to speed up the neural network training through distributed training offered by TensorFlow (Abadi et al., 2015). A **MirroredStrategy** was used for synchronized distributed training of the neural network models.

4.2 Forecasting problem and description

The forecasting problem was to produce a day-ahead forecast with a horizon of 24 hours and an hourly resolution. This is a multi-step ahead forecasting problem where the horizon

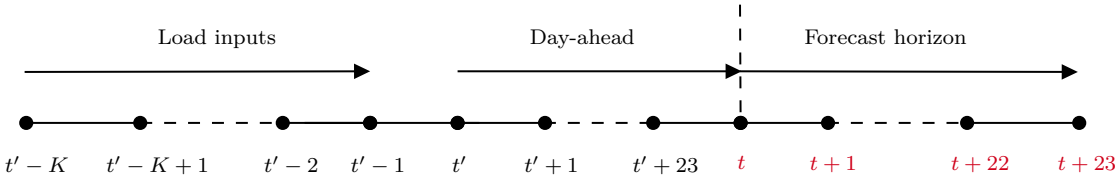


Figure 4.1: Timeline of the forecast strategy and power load. t' is the termin time of the forecast and t is the first forecast hour of power load.

is $H = 24$ and the lead time is $D = 24$ (see section 2.1.4). In essence, *historical* load data is being "forecast". Hence, it is crucial not to lose track of the limitations which would be present in a real forecasting scenario. I.e., the forecast needs to have been possible in a real world scenario. For starters, let the termin time of each forecast be at midnight each day (i.e., the forecast is run every day at 00:00). This also implies a forecasting interval of 24 hours ($I = 24$), and that the first forecast value is the one denoting the average electrical load of the hour 00-01 (see section 3.1).

The MIMO forecasting strategy (see equation 2.7) was applied for all the models apart from the MLR model. The reasoning behind this approach is threefold: (1) there are good results reported in the literature for this method (Ben Taieb et al., 2012); (2) the alternative was to use a direct or a recursive approach, where the former was deemed too computationally expensive and the latter may be inferior to MIMO; (3) only one model had to be evaluated for feature importance using XAI.

Let t' denote the termin time and $t = t' + D = t' + 24$ the first forecast hour (see figure 4.1). $\hat{P}^{(t)}$ then denotes the average electrical load of the first forecast hour one day ahead of the termin time. Generally, all the MIMO models may then be described by

$$\hat{\boldsymbol{P}} = f(\boldsymbol{P}, \boldsymbol{x}), \tag{4.1}$$

where $f : \mathbb{R}^n \to \mathbb{R}^{24}$; $\hat{\boldsymbol{P}} = \begin{bmatrix} \hat{P}^{(t)} & \hat{P}^{(t+1)} & \dots & \hat{P}^{(t+23)} \end{bmatrix}$ is the output vector of forecast values; $\boldsymbol{x}$ is the concatenated vector of other exogenous time series and variables; and $\boldsymbol{P} = \begin{bmatrix} P^{(t'-K)} & P^{(t'-K+1)} & \dots & P^{(t'-1)} \end{bmatrix}$ is the vector of historical load values. For the sake of clarity, and in the context of the midnight termin time, let d denote the forecast day and let $t = 0$, then the time lags of $\boldsymbol{P}$ may be referred to by their day number and hour of day

with respect to the forecast day. E.g., $\boldsymbol{P_{d-2}} = \begin{bmatrix} P_{d-2}^{(0)} & P_{d-2}^{(1)} & \dots & P_{d-2}^{(23)} \end{bmatrix}$ would in this case refer to the vector of historical power load two days prior to the forecast day. Similarly, $\hat{\boldsymbol{P}} = \hat{\boldsymbol{P}}_{\boldsymbol{d}} = \begin{bmatrix} \hat{P}_d^{(0)} & \hat{P}_d^{(1)} & \dots & \hat{P}_d^{(23)} \end{bmatrix}$ refers to the forecast power load.

4.3 Data preparation and pre-processing

This section covers the general pre-processing and data preparation which was done prior to adapting the data to each particular model.

4.3.1 Time series variables

The time series (continuous) variables which were included in the development phase of the thesis were:

(1) Historical electrical load, P

(2) Numerical weather predictions from ten locations

   (a) Temperature, $\hat{T}$

   (b) Humidity, $\hat{H}$

   (c) Pressure, $\hat{p}$

   (d) Wind speed, $\hat{v}$

   (e) Cloud cover, $\hat{c}$

The electrical load data and NWPs are described in detail in chapter 3. Apart from the preliminary data imputation of the data sets, the raw data of the continuous variables was generally prepared in the following manner: (1) split into training and test data sets; (2) normalized to the training data; and (3) reshaped to the expected input dimensions of the model. The split of training/test depended on the particular experiment and will be covered in later sections.

Normalization (see section 2.1.3) of ML models generally increases the model performance and stability, especially in the case of neural networks (Goodfellow et al., 2016). Both

standardization (equation 2.4) and min-max (equation 2.3) scaling was tested during development. Both of these were implemented using the Python libary sklearn's **StandardScaler** and **MinMaxScaler** (Pedregosa et al., 2011). Standardization outperformed min-max scaling most of the time during testing in development, so it was decided to keep using standardization. Sklearn's scalers supports online scaling, which was used to continuously update the scaler parameters as the forecasting models were trained on more samples over time.

Since time series forecasting is a time-dependent problem, not all historical data may be equally relevant. I.e., the underlying distribution of the variable may change over the years so that some older data is out-of-distribution. It was therefore attempted to use different sample weightings when fitting the scaler to the training data. For instance, it was attempted to only use the latest 1-3 years for scaling, but this did not seem to improve performance. A weighting based on exponential decay was also tested, where the weight α for tth sample was given by $\alpha^{(t)} = 1 - e^{-\frac{t}{N_{\mathcal{D}}}}$. This too did not improve the results. Regardless of the hypothetical soundness of scaling based on recency, none of these weighting strategies appeared to improve the results noticeably.

After scaling all the continuous data, the variables had to be reshaped into the number of features and shapes expected by the particular model. As described in section 2.2.1, data has to be reshaped into features and targets for a supervised learning problem. Since the raw data of the continuous variables are time series (i.e., only one feature), these have to be split into the appropriate shape expected by the particular model (i.e., to a shape of $samples \times n\_features$). For instance, the weather variables of each location (see section 3.2) were always reshaped to 24 features, one for each output hour of the forecast day, such that one sample (one day) of the predictions of a weather variable T at a particular location c may be given by $\hat{\boldsymbol{T}}_c = \begin{bmatrix} \hat{T}_c^{(t)} & \hat{T}_c^{(t+1)} & \dots & \hat{T}_c^{(t+24)} \end{bmatrix}$. The reader may want to refer to Brownlee (2018) for detailed illustrations regarding the transformations when going from time series data to supervised learning data.

4.3.2 Categorical variables

The categorical calendar variables of the forecasting models were implemented using one-hot encoding. I.e., given an ordinal categorical variable which takes on n distinct values (e.g., $d \in \{0, 1, 2, 3, 4, 5, 6\}$ representing Monday-Sunday), n one-hot encoded variables ($d_0, d_1, \ldots, d_6 \in \{0, 1\}$) were made. 0 and 1 represents the absence or presence of the respective encoded variable, e.g., $d_0 = 1$ indicating that Monday is "ON". Moreover, only *one* variable can be present at a time. The reasoning behind this approach instead of an ordinal approach is two-fold. Firstly, encoding each state of the original variable ensures that there is no bias involved with some states being represented by a larger numerical value than others, and hence having a larger impact on activations in a neural network; and secondly, by indicating absence or presence with 0 and 1, this allows for more intuitive feature attributions.

The calendar variables which were made for the power load forecasts were:

(1) Day of the week, $d_0, d_1, \ldots, d_6$

(2) Season of the year, s_0, s_1, s_2, s_3

(3) Month of the year, $m_0, m_1, \ldots, m_{11}$

(4) Holiday, h

(5) Easter, e

where the extents of s_0: winter, s_1: spring, s_2: summer and s_3: autumn were defined in the same manner as in section 3.1.2.

As discussed in section 3.1.3, the load profiles of some public holidays may only appear on that single day. They may also vary from year to year and are, to some extent, random. This makes public holidays difficult to model since there is limited amounts of data for each individual holiday. The holiday and Easter variables were therefore implemented so that h indicated the presence of a public holiday (see section 3.1.3) and e also indicated whether it was during Easter week or not. Easter week was defined as all days between Palm Sunday and Easter Monday. The reasoning for including both holidays and Easter variables, was

Table 4.1: The continuous and categorical variables

Category	Variable	Features
Continuous (time series)	Historical load	$P^{(t'-K)}, P^{(t'-K+1)}, \ldots, P^{(t'-1)}$
	Temperature	$\hat{\boldsymbol{T}}_{\boldsymbol{D}}, \hat{\boldsymbol{T}}_{\boldsymbol{F}}, \hat{\boldsymbol{T}}_{\boldsymbol{Hl}}, \hat{\boldsymbol{T}}_{\boldsymbol{Hm}}, \hat{\boldsymbol{T}}_{\boldsymbol{Ho}}, \hat{\boldsymbol{T}}_{\boldsymbol{Kb}}, \hat{\boldsymbol{T}}_{\boldsymbol{Kv}}, \hat{\boldsymbol{T}}_{\boldsymbol{L}}, \hat{\boldsymbol{T}}_{\boldsymbol{M}}, \hat{\boldsymbol{T}}_{\boldsymbol{O}}$
	Humidity	$\hat{\boldsymbol{H}}_{\boldsymbol{D}}, \hat{\boldsymbol{H}}_{\boldsymbol{F}}, \hat{\boldsymbol{H}}_{\boldsymbol{Hl}}, \hat{\boldsymbol{H}}_{\boldsymbol{Hm}}, \hat{\boldsymbol{H}}_{\boldsymbol{Ho}}, \hat{\boldsymbol{H}}_{\boldsymbol{Kb}}, \hat{\boldsymbol{H}}_{\boldsymbol{Kv}}, \hat{\boldsymbol{H}}_{\boldsymbol{L}}, \hat{\boldsymbol{H}}_{\boldsymbol{M}}, \hat{\boldsymbol{H}}_{\boldsymbol{O}}$
	Pressure	$\hat{\boldsymbol{p}}_{\boldsymbol{D}}, \hat{\boldsymbol{p}}_{\boldsymbol{F}}, \hat{\boldsymbol{p}}_{\boldsymbol{Hl}}, \hat{\boldsymbol{p}}_{\boldsymbol{Hm}}, \hat{\boldsymbol{p}}_{\boldsymbol{Ho}}, \hat{\boldsymbol{p}}_{\boldsymbol{Kb}}, \hat{\boldsymbol{p}}_{\boldsymbol{Kv}}, \hat{\boldsymbol{p}}_{\boldsymbol{L}}, \hat{\boldsymbol{p}}_{\boldsymbol{M}}, \hat{\boldsymbol{p}}_{\boldsymbol{O}}$
	Wind speed	$\hat{\boldsymbol{v}}_{\boldsymbol{D}}, \hat{\boldsymbol{v}}_{\boldsymbol{F}}, \hat{\boldsymbol{v}}_{\boldsymbol{Hl}}, \hat{\boldsymbol{v}}_{\boldsymbol{Hm}}, \hat{\boldsymbol{v}}_{\boldsymbol{Ho}}, \hat{\boldsymbol{v}}_{\boldsymbol{Kb}}, \hat{\boldsymbol{v}}_{\boldsymbol{Kv}}, \hat{\boldsymbol{v}}_{\boldsymbol{L}}, \hat{\boldsymbol{v}}_{\boldsymbol{M}}, \hat{\boldsymbol{v}}_{\boldsymbol{O}}$
	Cloud cover	$\hat{\boldsymbol{c}}_{\boldsymbol{D}}, \hat{\boldsymbol{c}}_{\boldsymbol{F}}, \hat{\boldsymbol{c}}_{\boldsymbol{Hl}}, \hat{\boldsymbol{c}}_{\boldsymbol{Hm}}, \hat{\boldsymbol{c}}_{\boldsymbol{Ho}}, \hat{\boldsymbol{c}}_{\boldsymbol{Kb}}, \hat{\boldsymbol{c}}_{\boldsymbol{Kv}}, \hat{\boldsymbol{c}}_{\boldsymbol{L}}, \hat{\boldsymbol{c}}_{\boldsymbol{M}}, \hat{\boldsymbol{c}}_{\boldsymbol{O}}$
Categorical (calendar)	Day of the week	$d_0, d_1, \ldots, d_6$
	Season	s_0, s_1, s_2, s_3
	Month of the year	$m_0, m_1, \ldots, m_{11}$
	Holiday	h
	Easter	e

that Easter (see figure 3.6) appeared to lower load over the entire week of Easter, and not just on the official holidays.

Table 4.1 shows all the continuous and categorical variables which were considered for the forecasting models. Their structure are also shown.

4.3.3 Daylight saving time

Recall from section 3.1.4 that the electrical load follows local time and hence also experiences DST. If this is not dealt with during data pre-processing, the model forecasts will lag the actual load by an hour from the last Sunday in March until the last Sunday in October. To remove this effect, one hour was removed at 2AM on the last Sunday in March, and one hour was duplicated at 2AM on the last Sunday in October across the data set.

4.4 Evaluation of forecasting accuracy

In order to evaluate the forecasting accuracy of a model, some sort of metric has to be chosen. Every metric has its own set of limitations where a model may perform well based on the

results from one metric, but poor according to another. For instance, consider a model which is very accurate most of the time, but in rare occasions gives completely wrong forecasts. A metric which measures the general forecasting performance of the model would give this model a good score, while a metric which penalizes large deviations would give the model a worse score, depending on how much deviations are penalized.

Two such popularly used metrics are the mean absolute percentage error (MAPE) and root mean squared error (RMSE). The MAPE of some set of predictions is given by

$$MAPE = \frac{1}{N}\sum_{i=0}^{N-1}\frac{|P^{(i)} - \hat{P}^{(i)}|}{P^{(i)}} \cdot 100\%, \tag{4.2}$$

while the RMSE of the same set of predictions is given by

$$RMSE = \sqrt{\frac{\sum_{i=0}^{N-1}(P^{(i)} - \hat{P}^{(i)})^2}{N}}, \tag{4.3}$$

where N are the number of predictions to be evaluated for the particular set.

Note that MAPE and RMSE does not give any direct indication as to how well the model was able to capture the load profiles of power load (see section 3.1.2). In order to evaluate this, visual investigation has to be performed. This was done for some of the models during development to validate that the models were in-fact learning the correct shapes of power load variation. Visual confirmation is also a way of judging if the models are over-fitting or under-fitting the data. I.e., if the model in question is trained sufficiently long, but the forecasts only resemble a simplistic power load shape, then this may be a sign that the model's representational capacity is too low. Conversely, if the forecast shapes appear jittery and noisy, this may be a sign that the model is over-fitting.

4.5 Development of the candidate models

The first part of the thesis involved the development of a suitable load forecasting model. During development, the available data was constrained to 2014-2019, leaving out the final year for later evaluation. 2019 was used as the test year for all models during development,

while the years of 2014-2018 were used for training data. In a real world forecasting scenario, newly observed samples are continually made available as time progresses. To keep the models up to date, they have to be re-trained or updated with these new samples. It was chosen to use an updating cycle of one week for the models during development testing. I.e., the models were tested by simulating one forecast each "day" at 00:00, and re-training all the models with the seven samples of the test set which had been "observed". This meant that 52 different versions of every model were trained for every development evaluation of the test set. The parameters of all the model versions were saved so that they could be used later without having to re-do the evaluation. This was also convenient because when explaining a forecast with SHAP later on, the appropriate model and forecast could be loaded from memory. Different updating cycles were attempted, but the difference between an updating cycle of, say, one day versus seven days was negligible. The difference in performance between updating the model as time progressed versus just training *one* model at the start of 2019 was noticeable, but to the author's surprise it was not very big. This may be due to the fact that the power load of NO1 has been seemingly stable for the observed time period (see section 3.1).

4.5.1 Baseline models

The baseline models were selected as references for the performance of the neural network models. Baseline models are usually more crude than the proposed models, hence it is expected that they will be outperformed by the other models. If they are not, it may suggest that there is something wrong with the proposed models. It may also mean that there is little point in using a more complex model for a problem that is sufficiently modelled by a simpler and more interpretable model.

A Naive model, also commonly called a persistence model, is often used as a baseline for time series with strong seasonality (like power load). The Naive forecast assumes the day-ahead values to be exactly the same as some prior day. Usually, the observed power load values of the day before $\boldsymbol{P}_{d-1}$ are used, since these are the closest to the forecast time. However, since the forecast horizon is day-ahead, the latest available values at termin time are the $\boldsymbol{P}_{d-2}$

observed values. Another option is to use the power load of the same day of the previous week, $\boldsymbol{P}_{d-7}$. Given the weekly seasonality of power load (see section 3.1.2) this may not be a bad assumption. Using only $\boldsymbol{P}_{d-2}$, the Naive forecast model takes on the shape of equation 4.1, and may be formulated as

$$\hat{\boldsymbol{P}} = \boldsymbol{P}_{d-2}, \tag{4.4}$$

which served as the first baseline model. The latest $\boldsymbol{P}_{d-2}$ were chosen over $\boldsymbol{P}_{d-7}$ because despite seemingly good results during stable periods (summer in particular), the power load was not as consistent during other periods (see spring and autumn).

The second chosen baseline model was a direct MLR (see section 2.2.2) model using lags of the historical power load as inputs. By using the PACF plot in figure 3.5 and experimenting with different lags, it was found that historical load a week back from the termin time was sufficient. This type of direct forecasting model (see equation 2.6) may be given by

$$\hat{P}^{(t+h)} = f_h(\boldsymbol{P}_{d-2}, \boldsymbol{P}_{d-3}, \boldsymbol{P}_{d-4}, \boldsymbol{P}_{d-5}, \boldsymbol{P}_{d-6}, \boldsymbol{P}_{d-7}, \boldsymbol{P}_{d-8}) = \boldsymbol{P}\boldsymbol{\beta}_{\boldsymbol{h}}, \tag{4.5}$$

where $h \in [0, 23]$ is the target output hour and $\boldsymbol{\beta}_{\boldsymbol{h}} \in \mathbb{R}^{168}$; f_h represents the MLR model pertaining to the hth output hour. I.e., there were twenty-four different MLR models with twenty-four different regression coefficient vectors. These models were implemented using sklearn's **MultiOutputRegressor** and **LinearRegression** classes (Pedregosa et al., 2011). The collective forecast made by these models is referred to as MLR-1.

The MLR model was chosen as one of the baselines to compare the relatively complex neural network models to a simple one. However, in the end not much time was spent on developing the MLR model. Hence, it may have suffered from lack of feature-engineering, such as not hand-picking power load lags; not including trend or seasonality components; and not investigating interaction effects between lags, weather variables and time components. For a more comprehensive development process of a MLR model for electrical load forecasting, the reader is referred to Hong (2010).

4.5.2 Neural network models

All the MLPs and CNNs were built using the Keras (Chollet et al., 2015) open-source library in Python. Keras offers high-level implementations for the neural network architecture and optimization. This means that most layers, activation functions, optimizers and cost functions are readily available and easy to implement with the use of Keras. Particularly, one-dimensional convolutions (see section 2.3.4) and fully-connected layers (see section 2.3.1) were implemented with the Keras **Conv1D** and **Dense** layers. MSE (see equation 2.10) was used as the cost function for all the neural network models. For regularization (see section 2.3.6), **Dropout** layers from Keras were added after every **Dense** layer. To reduce feaure map dimensions **MaxPooling1D** layers were added after each **Conv1D** layer.

It is common-practice to use about 20% of the training data as validation data. However it was found that about 15% of randomly-picked training samples were sufficient for early-stopping validation data. All models were trained with a batch size of 32 and 200 as the maximum number of epochs, but were generally stopped by early-stopping before reaching 150 epochs.

Network architecture and hyper-parameter optimization

The network architecture and its hyper-parameters (hidden layers, hidden units, activation functions, filters, etc.) determines the model's prediction capacity and has to be found on a case-by-case basis since every forecasting problem is different. It is desirable that the network is able to represent the underlying mapping of features to targets as well as possible, without biasing too heavily towards the training data (over-fitting). A theoretically optimal architecture for one set of features is not guaranteed to perform well given a different set of features. Likewise, a good architecture for power load forecasting may change over the years and depends on the data given to the model.

Hyper-parameter optimization is the procedure of optimizing the architecture and all the hyper-parameters of the model to the problem in question. Three frequently used methods include manual searches, grid searches and random searches (Bergstra and Bengio, 2012). While grid searches guarantee to find the best combination within the search space (by iterat-

ing through every single possible combination), the number of different combinations grows exponentially with increasing number of hyper-parameters. Hence, Bergstra and Bengio (2012) argue that random searches are superior to grid searches both in terms of efficiency and performance. Hyper-parameter optimization for all the neural network models was therefore done with a combination of heuristics and random searches of the hyper-parameter space. I.e., the search space was initially determined and modified by trial-and-error and common recommendations found in the literature. Random searches were then used to search this hyper-parameter space and find a satisfactory combination for the particular model on a validation set. It is important to note that the hyper-parameter optimization procedure was a continued work-in-progress. Hence, later models might have benefit from the experience gained from trial-and-error of earlier models.

Implementation of the random search was done using the **RandomSearch** class from Keras Tuner (O'Malley et al., 2019). The search space was defined through a custom class which inherits from Keras Tuner's **HyperModel** class. During model development, as more features were added and the models changed, new architectures and hyper-parameters had to be established. A more detailed description of the hyper-parameter search methodology; all the model architectures; and details on optimizers and hyper-parameters are given in appendix B.

First generation: historical power load

The first neural network models in development were endogenous models. I.e., they were models using only the observed power load as inputs. The same number of power load lags as MLR-1 were used. The forecasts of these models may then be given by

$$\hat{\boldsymbol{P}} = f(\boldsymbol{P}_{d-2}, \boldsymbol{P}_{d-3}, \boldsymbol{P}_{d-4}, \boldsymbol{P}_{d-5}, \boldsymbol{P}_{d-6}, \boldsymbol{P}_{d-7}, \boldsymbol{P}_{d-8}), \tag{4.6}$$

where $f : \mathbb{R}^{168} \to \mathbb{R}^{24}$ is a vector-valued function representing the particular model.

The first generation of neural network models were made up of a CNN and an MLP, which are referred to as CNN-1 and MLP-1. The architectures of these two models are shown in figures B.2 and B.1; their hyper-parameters are listed in tables B.4 and B.3. This particular

generation of models were compared to the baseline models and the best performing model was chosen for the next generation. Ideally, all the different models would have been further developed for all generations, but due to time constraints this was not done.

Second generation: meteorological variables

The best candidate from the first generation models was CNN-1. It was also decided to continue working with this type of neural network due to its special ability to handle long sequences of correlated, multi-dimensional data (see section 2.3.4). In particular, recall that each meteorological variable described in section 3.2 is represented by ten different locations to choose from. Deciding how to attribute importance to each location is a difficult problem given the large territory covered by the NO1 price region. However, a similar problem found in the literature, is the weather station selection problem for historical weather data (Hong et al., 2015). A commonly used selection method for this problem is to use a weighted linear combination of the different locations and then find the weights through optimization. Instead of using linear combinations, using convolutional layers to find the optimal weight combination is essentially the same idea, except it also allows for non-linear combinations. Spatial feature extraction of NWPs using convolutional layers has been done for wind and solar energy forecasting (Higashiyama et al., 2018; Chen et al., 2017), but to the author's knowledge there are no examples of this in the load forecasting literature.

Hence, the method for determining the weights of each location was to let the neural network find the weights through multi-channel convolutional layers, where the channel axis represented the different locations. To illustrate this concept, consider the first element of the jth temperature feature map produced by the jth filter with unit stride, valid padding

and a kernel size $N_k > 2$. From equation 4.7, this is given by

$$\begin{aligned} z_j^{(0)} = \sum_{c \in \mathcal{C}} \hat{\boldsymbol{T}}_c^{(0:0+N_k-1)} \boldsymbol{w}_{c,j}^T = w_{D,j}^{(0)} \hat{T_D}^{(0)} + w_{D,j}^{(1)} \hat{T_D}^{(1)} + \ldots + w_{D,j}^{(N_k-1)} \hat{T_D}^{(N_k-1)} + \\ w_{F,j}^{(0)} \hat{T_F}^{(0)} + w_{F,j}^{(1)} \hat{T_F}^{(1)} + \ldots + w_{F,j}^{(N_k-1)} \hat{T_F}^{(N_k-1)} + \\ \vdots \\ w_{O,j}^{(0)} \hat{T_O}^{(0)} + w_{O,j}^{(1)} \hat{T_O}^{(1)} + \ldots + w_{O,j}^{(N_k-1)} \hat{T_O}^{(N_k-1)}, \end{aligned} \tag{4.7}$$

where $\mathcal{C}$ is the set of locations; $\hat{\boldsymbol{T}}_c^{(0:0+N_k+1)}$ is the slice of the temperature predictions in location c with size N_k; and $\boldsymbol{w}_{c,j}$ is the kernel pertaining to the particular city and filter. It can be seen that the result of the convolution are weighted sums of the NWP locations and the time steps of the receptive field elements. The kernel size decides how many time steps are linearly combined for each element of the output feature map. The same sort of weighted sum of adjacent time steps can be shown for the historical power load.

The idea behind this sort of approach is that the model can find the optimal combinations through optimization. Since adjacent time steps are highly correlated, using a convolutional layer over a fully-connected layer is preferred in order to prevent over-fitting and to reduce the number of parameters. Additionally, it can be shown that aggregate features, like average, maximum and minimum of a window, can be produced by particular filter arrangements (Bailey, 2013). Note that by using a multi-channel approach, **Conv1D** expects each input to be a rank three tensor $\hat{\boldsymbol{T}}_c \in \mathbb{R}^{N \times L \times N_c}$, where $N \times L \times N_c = N \times 24 \times 10$; N are the number of samples; $L = 24$ is the sample length; and $N_c = 10$ are the number of locations (see section 2.3.4).

Temperature predictions are the most important for power load forecasting. Hence the first second generation model (CNN-2-1) was only given temperature as an input. This was done by adding another convolutional branch to the model and concatenating the power load and the temperature feature maps (see appendix B). This model may be described by

$$\hat{\boldsymbol{P}} = f(\boldsymbol{P}, \boldsymbol{T}), \tag{4.8}$$

where $f : \mathbb{R}^{408} \rightarrow \mathbb{R}^{24}$ and $\boldsymbol{T}$ is a tensor of temperature predictions for all ten locations, where the shape of one input is $1 \times 24 \times 10$. The architecture of CNN-2-1 is shown in figure B.3 and its hyper-parameters are listed in table B.5

The second model of the second generation (CNN-2-2) was given all the weather variables from table 4.1 as inputs to see if this would improve performance. Each weather variable was given its own convolutional branch, and all the resulting feature maps were concatenated after pooling layers in each branch. This model may be described as

$$\hat{\boldsymbol{P}} = f(\boldsymbol{P}, \boldsymbol{T}, \boldsymbol{H}, \boldsymbol{p}, \boldsymbol{v}, \boldsymbol{c}), \tag{4.9}$$

where $f : \mathbb{R}^{1368} \rightarrow \mathbb{R}^{24}$; and $\boldsymbol{T}$ $\boldsymbol{H}$, $\boldsymbol{p}$, $\boldsymbol{v}$ and $\boldsymbol{c}$ denote temperature, humidity, pressure, wind and cloud cover tensors, respectively. The architectures of CNN-2-2 is shown in figure B.4 and its hyper-parameters are listed in table B.6

The third model developed during the second generation (CNN-2-3) was a model with temperature, humidity and pressure inputs. Hence

$$\hat{\boldsymbol{P}} = f(\boldsymbol{P}, \boldsymbol{T}, \boldsymbol{H}, \boldsymbol{p}), \tag{4.10}$$

where $f : \mathbb{R}^{888} \rightarrow \mathbb{R}^{24}$. The architecture of CNN-2-3 is shown in figure B.5 and its hyper-parameters are listed in table B.7

Third generation: calendar variables

The third generation models were given the calendar variables from table 4.1. The first model (CNN-3-1) may be described by

$$\hat{\boldsymbol{P}} = f(\boldsymbol{P}, \boldsymbol{T}, \boldsymbol{H}, \boldsymbol{p}, \boldsymbol{d}, \boldsymbol{s}, \boldsymbol{m}, h, e), \tag{4.11}$$

where $f : \mathbb{R}^{913} \rightarrow \mathbb{R}^{24}$; $\boldsymbol{d} = \begin{bmatrix} d_0 & d_1 & \dots & d_6 \end{bmatrix}$ is the vector of day of the week features; $\boldsymbol{s} = \begin{bmatrix} s_0 & s_1 & s_2 & s_3 \end{bmatrix}$ is the vector of season features; and $\boldsymbol{m} = \begin{bmatrix} m_0 & m_1 & \dots & m_{11} \end{bmatrix}$ is the vector of month of the year features. The second model (CNN-3-2) may be described similarly, but

with changes to the implementation of the holiday and Easter features (described in chapter 5). The architecture and hyper-parameters of CNN-3-1 and CNN-3-2 are shown in figure B.6 and table B.8.

4.6 Deployment and testing of the selected model

In this part of the thesis, the final model's (CNN-3-2) capabilities were tested on previously unseen data. In other words, the year of 2020, which was left out during development, was made available for testing the final model. Additionally, tests like the expanding window and rolling window tests were performed to get an idea of the model's generalization performance (see section 2.2.1). Keep in mind however, that since data from 2014-2019 was used during model development, the forecast performances outside of the hold-out year of 2020 have to be considered with scrutiny as they may have been subject to look-ahead bias.

The same updating cycle as in the development phase (see section 4.5) was used for all the tests. I.e., during the test year, the model was re-trained every week, resulting in 52 different models for each test. Note that the same hyper-parameters which were found for CNN-3-2 were used in all the tests. Randomly-picked 15% of the training data was again also used as early-stopping validation data.

4.6.1 Expanding window

To test a model's generalization capabilities test strategies like a K-fold cross-validation is common (Hyndman and Athanasopoulos, 2018). However, this type of test cannot be used for time series data since it would be prone to look-ahead bias. There are various other ways, particular to time series problems, which test the capability of a forecasting model. An expanding window test uses all available information prior to the particular test year, where the number of test years depend on the specified minimum amount of training years. The minimum number of training years was set to three years. In this case, the first test used 2014-2016 as training data and 2017 as the test year; the second test used 2014-2017 as training data and 2017 as the test year; and so on. The expanding window test and the four test folds are illustrated in table 4.2.

Table 4.2: Expanding window test strategy

	2014	2015	2016	2017	2018	2019	2020
Test 1	Train			Test			
Test 2	Train				Test		
Test 3	Train					Test	
Test 4	Train						Test

4.7 Interpretation of forecasts

The Deep SHAP estimation method for SHAP values (see section 2.4.3) was implemented using the SHAP open-source Python library and its **DeepExplainer** class (Lundberg and Lee, 2017). The explainer works for both Keras and PyTorch neural networks. The class takes the model instance and a background set in order to calculate SHAP values for a specified set of samples. SHAP was chosen over other XAI methods for three reasons: (1) explanations have been shown to be closely aligned with human intuition (Lundberg and Lee, 2017); (2) computations of SHAP values with Deep SHAP are very fast; (3) the framework of SHAP is based on Shapley values and its axioms, which inspires confidence in its theoretical soundness.

Because of the very large number of time series features in the models (see table **??**), the SHAP values of the continuous variables were aggregated into *one* SHAP value for every twenty-four time steps of every variable. I.e., the twenty-four values of every weather variable and every location were aggregated so that

$$\phi_{\hat{\boldsymbol{x}}_c} = \sum_{i=1}^{24} \phi_{\hat{x}_c^{(t+i)}}, \tag{4.12}$$

where x is the particular weather variable and c is the location. And similarly for the SHAP values of power load:

$$\phi_{\boldsymbol{P}_{d-k}} = \sum_{i=1}^{24} \phi_{P_{d-k}^{(i)}}, \tag{4.13}$$

where $k \in [2, 8]$ denotes the particular day of power load lags.

Since $\phi \in \mathbb{R}$, SHAP values can take on both negative and positive values. Hence, importance of a variable may potentially be lost by terms cancelling each other out. To prevent this from happening, *absolute* SHAP values may be aggregated instead. Then equation 4.12 and 4.13 are slightly modified so that

$$|\phi_{\hat{\boldsymbol{x}}_c}| = \sum_{i=1}^{24} |\phi_{\hat{x}_c^{(t+i)}}|, \tag{4.14}$$

and

$$|\phi_{\boldsymbol{P}_{d-k}}| = \sum_{i=1}^{24} |\phi_{P_{d-k}^{(i)}}|. \tag{4.15}$$

Note that by using absolute SHAP values, property (1) of SHAP is violated. I.e., the SHAP values will no longer add up to the difference in prediction from the baseline. Furthermore, keep in mind that each SHAP value is particular to each output of the model.

4.7.1 Choosing background data

Choosing the right background data to represent missing features is an important part of the XAI model and the subject of an ongoing debate. As Sturmfels et al. (2020) note in their comprehensive discussion on the choice of background data, the difficulty in evaluating different background choices remains that although an explanation may seem reasonable, there is no "ground truth" explanation of how the model works.

Initially, the default recommendation to randomly sample the background set from the training data was used for all explanations. However, as Shrikumar et al. (2019) and Sturmfels et al. (2020) note, the choice of baseline is crucial to the intuitiveness of the explanation and has to be well thought out. Particularly, domain knowledge can be leveraged to choose intuitive baselines. In the case of load forecasting, instead of randomly sampling from the training set, choosing baselines based on the periodic nature of power load might give more insightful results. E.g., consider a conspicuous load forecast for a Tuesday in the middle of summer, when it is known that summers are usually very stable (see section 3.1.2). Intuitively it makes more sense to compare the model's forecast to a previously forecast Tuesday during summer, or several Tuesdays. The obvious advantage of this choice of baseline is that

it is more in line with the form of reasoning a human might use. One potential disadvantage is that the amount of samples may be too few to estimate SHAP values. Moreover, if the model uses calendar features, the impact of these features may be ignored due to there being no difference to the baseline.

The above reasoning holds for *local* explanations (i.e., for an hour or a day of forecasts), when debugging or trying to understand a few selected forecasts. When using SHAP for feature selection and an over-all understanding of the model however, *global* explanations (using all forecasts) of the model's inner workings are needed to see which features are important *most* of the time. Hence, two fundamentally different types of explanations were experimented with during development and deployment: (1) global explanations using the training data as background data and all forecasts as samples; and (2) local explanations using hand-picked background samples.

4.7.2 Global explanations for feature selection

During the development of the neural network models, global explanations of feature importance using Deep SHAP were made. Global SHAP feature importance was produced by considering the average effect a feature had over the test set, and averaging this effect over the output hours of the model. I.e., let the average SHAP value of some variable $\boldsymbol{x}$ be given by

$$\bar{\phi}_{\boldsymbol{x}} = \frac{1}{24N_{\mathcal{T}}} \sum_{h=0}^{23} \sum_{i=0}^{N_{\mathcal{T}-1}} |\phi_{\boldsymbol{x}}(h,i)|, \tag{4.16}$$

where $N_{\mathcal{T}}$ are the number of samples in the test set; and $|\phi_{\boldsymbol{x}}(h,i)|$ is the absolute SHAP value (equation 4.14 and 4.15) of the variable $\boldsymbol{x}$ for the hth hour and ith sample. Note that the *absolute* SHAP value for each variable is used for calculation of global importance. This was done in order to prevent terms from cancelling each other out, which could have lead to a variable being attributed less importance than it should.

To calculate average SHAP values of each feature, the background data may be sampled from the training data. As a consequence, the SHAP value of a feature for a particular hour and instance may be interpreted as the average contribution the sample had over the

training data.

4.7.3 Local explanations for interpretation

In contrast to global explanations, the local explanations were generated based on the idea that recent, similar forecasts may provide more intuitive references in a time series problem than sampling from the training set. It is important to note that this is speculation, and that the mathematical rigorousness of this assumption is not known. I.e., if this choice violates any of the SHAP properties by not sampling from the distribution of expected values.

From equation 2.49 it can be seen that the sum of features of an instance contribute by $f(\boldsymbol{x}) - E[f(\boldsymbol{x})] = \sum \phi$. Hence, local explanations may be interpreted as "Feature i contributed ϕ_i to the difference from the expected prediction $f(\boldsymbol{x}) - E[f(\boldsymbol{x})]$". I.e., due to some difference in input $\Delta x_i = x_i - E[x_i]$, x_i contributed ϕ_i to push the prediction away from the expected prediction.

Chapter 5

Results and discussion

In this chapter the most important results from the models described in chapter 4 are presented, along with an analysis of the electrical load forecasts. First, the results of the models which were developed are presented and analysed; then the last model is put in deployment and the results are shown and analysed; lastly, the implications of the results in reference to the thesis goals are discussed.

Note that the full results of the held out year (2020) have been left out for the sake of brevity and can be found in its entirety in appendix C.

5.1 Model development

The first results of the thesis are those generated during model development. The results generated in this phase can be split into two: (1) the forecast improvements, both numerically and visually, with each model generation (see section 4.5.2); and (2) the interpretations made using Deep SHAP (see section 4.7) and its implications.

5.1.1 First generation models

As described in section 4.5.2, the first models only had past power load one week back as inputs. The results of the development year (2019) for these models are shown in tables 5.1 and 5.2. Table 5.1 shows the MAPE and RMSE (see section 4.4) of the models for each

Table 5.1: Metrics for the endogenous models and the Naïve forecast given by MAPE/RMSE.

Model	Winter		Spring		Summer		Autumn		Full year	
MLP-1	**5.56**	**386**	6.61	326	5.05	181	**4.61**	**249**	5.46	**293**
CNN-1	5.70	402	**6.59**	**317**	**4.39**	**159**	4.72	252	**5.35**	295
MLR-1	5.85	418	7.41	359	5.95	213	4.85	265	6.02	321
Naive	8.68	612	10.3	503	9.85	369	9.16	500	9.53	500

season and the full test year. Table 5.2 shows the MAPE and RMSE calculated by day of the week. The best model for each season and metric is outlined in bold.

As expected, the Naive model performs quite poor compared to the other three models. Usually, when the Naive forecast is based on the $d-1$ observed power load, the results are much better. However, since the Naive model was based on the only available observations at termin time ($\boldsymbol{P}_{d-2}$ observations of the load), the results are not very good. Out of the three ML models, CNN-1 appears to perform the best on the test year, but is outperformed by MLP-1 in Winter and Autumn. MLR-1 is surprisingly competitive with the other models, which may be attributed to the limited amount of information contained in the historical load input. I.e., that there is only so much accuracy possible to achieve with an endogenous model. In general, it appears that spring is the hardest season to forecast, which is intuitive since the weather may change rapidly during this season and the models have no way of knowing. Winter also appears to be more challenging than Summer and Autumn, likely due to the same reasons as for Spring.

From table 5.2 it can be seen that the MLP-1 and CNN-1 models outperform MLR-1 on all days of the week. Interestingly however, Friday and Wednesday have the worst performance across all models. Since the models have no calendar inputs (e.g., day of the week or month), it was expected that they would perform worse for the weekend due to the dissimilarity to weekdays. As investigated in chapter 3.1.2, weekdays are generally very similar in load profile, while the weekend has slightly different characteristics. However, Friday is also the weekday which deviates the most from the load profiles of the other weekdays. This may explain why the models are having the most trouble with Fridays.

Table 5.2: Metrics by day of the week for the first generation models in MAPE/RMSE.

Model	Monday		Tuesday		Wednesday		Thursday		Friday		Saturday		Sunday	
MLP-1	**4.54**	**247**	5.45	**322**	6.32	353	5.07	285	6.47	341	**5.35**	**262**	4.99	**215**
CNN-1	4.93	274	**5.42**	323	**6.11**	**343**	**4.98**	**273**	**6.03**	**323**	5.48	284	**4.51**	231
MLR-1	5.14	278	6.21	353	6.73	390	5.65	332	6.64	355	6.25	284	5.51	224

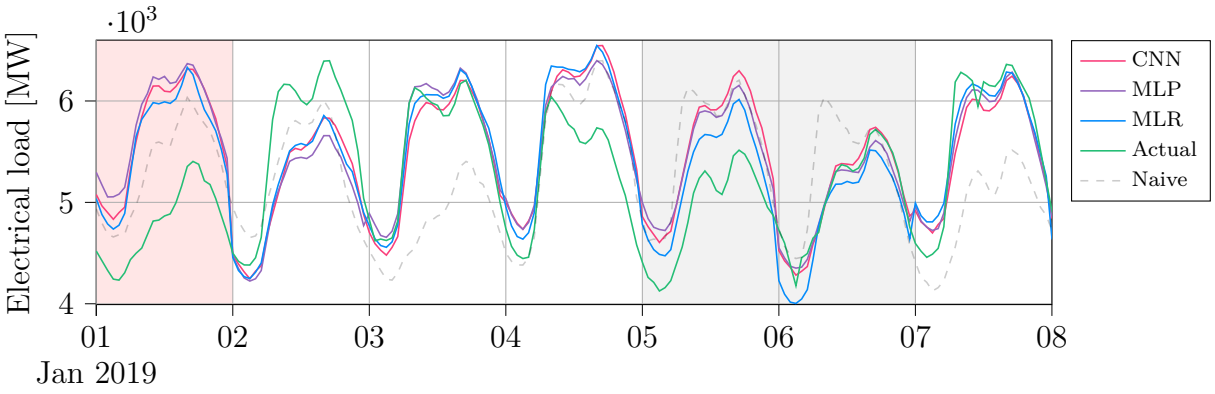


Figure 5.1: The first generation forecasting results for the first week of 2019.

Figure 5.1 shows the visual results for the first week of forecasts. As in chapter 3.1.2, the weekend has been highlighted with gray. Likewise, the background of New Year's Day has been highlighted with red. The above remarks regarding weekday performance may also be observed from the figure; i.e., all models have trouble with the Wednesday and Friday. There is also a large error on New Year's Day, since the first generation of models have no way of knowing whether the day-ahead forecast is on a holiday or not. However, the models do appear to have learned the weekly cycle of power load. The forecasts of the two months with worst and best performance can be found in appendix B.

Due to the large number of lags, it may be hard to judge the importance of each individual lag in the three ML models. Recall from section 4.5.1 that every output hour prediction from MLR-1 is a linear combination of the power load lags. This means that the coefficient of a lag j with respect to the particular output hour h, directly tells how much it contributed to the prediction. I.e., if the lag $P^{(t'-j)}$ changes by some amount $\Delta P^{(t'-j)}$ then the forecast for that hour changes by $\Delta \hat{P}^{(t+h)} = \beta_{hj} \Delta P^{(t'-j)}$. Note the distinction of the termin time t' and the first forecast hour t (refer to section 4.2 for details). Figure 5.2 shows a heatmap of the

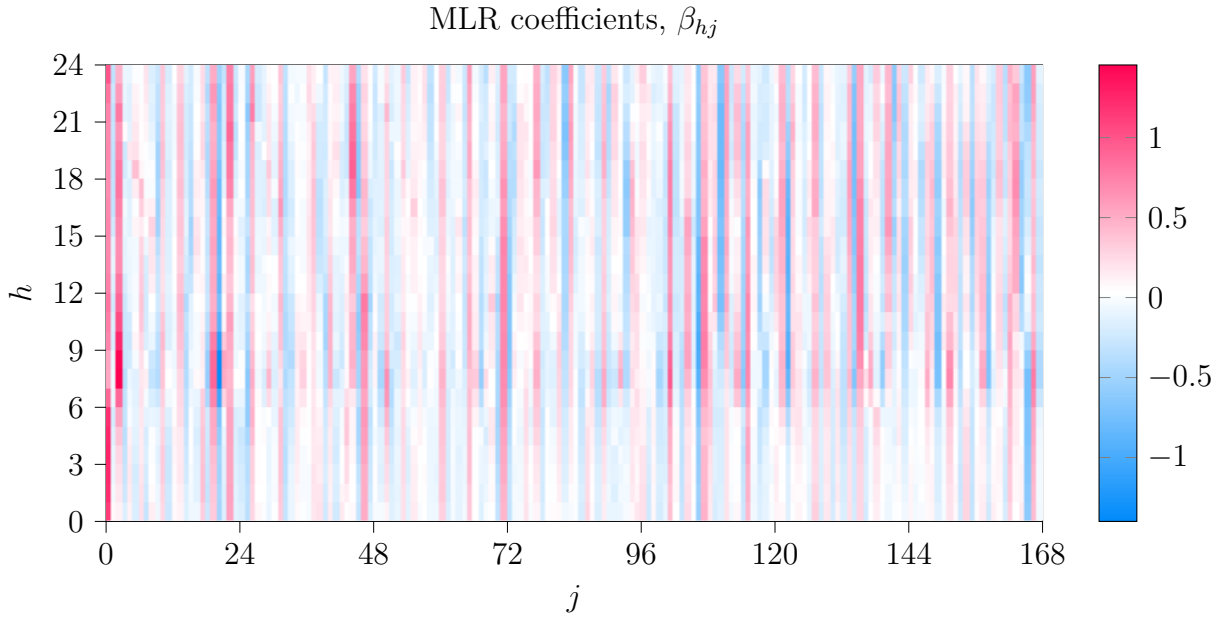


Figure 5.2: Heatmap of the MLR coefficients for each model output hour.

MLR-1 coefficients with respect to the output hour h and lag j. Note that the coefficients pertain to the final model of the test. I.e., the 52nd, since each ML model was re-trained every week.

Some observations from the heatmap can be made regarding lag importance. Firstly, the coefficients of MLR-1 are very similar for every output hour, as seen by the vertical stripes; secondly, the first and third lags have large coefficients for some of the hours, which implies that the newest observations are the most important; and thirdly, the most vibrant stripes can be seen around multiples of 24, which is consistent with the results from the PACF plot from section 3.1.2. Many of the lags between multiples of 24 are given little importance, since they are very dim for every output hour.

Interpreting the importance of each power load lag for CNN-1 and MLP-1 is more difficult for two reasons: (1) both these models consist of multiple layers of many non-linearized combinations of the inputs, hence there is no *direct* way, as above, to measure the contribution of each input; and (2) XAI methods, like Deep SHAP, only satisfy its axioms when used on one forecast at a time. Hence, information may be lost when SHAP values are averaged over many instances. This is in contrast to MLR-1 where the coefficients remain relatively

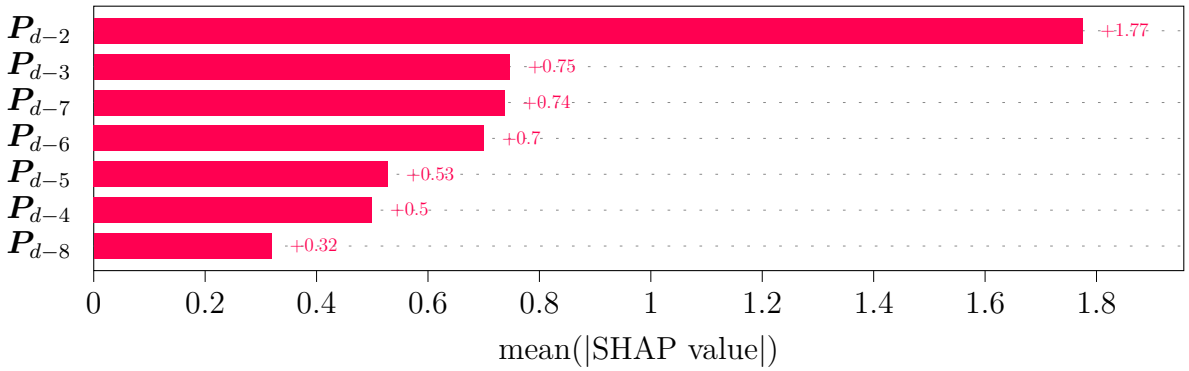


Figure 5.3: Global feature importance for CNN-1. Each SHAP value is the average across output hours and instances.

constant across all forecasts. Regardless, a global explanation for CNN-1 was produced and is shown in figure 5.3. As explained in section 4.7.1, sampling from the training set for background data is appropriate for a global explanation and was selected for the explanation. Moreover, each SHAP value was produced by: (1) using equations 4.14 and 4.15 to find the absolute contribution of each power load day of lags, and then (2) using equation 4.16 to average the absolute SHAP values over instances and output hours. Note therefore that each value is the *average* absolute contribution of each day of power load lags compared to typical values of the background data. As with the MLR heatmap, the final model of CNN-1 was used to produce the global explanation.

As can be seen from figure 5.3, Deep SHAP attributed the most importance to the power load lags two days prior to the forecast day, while lags three and seven days were attributed about the same and second-largest importance. This is also consistent with the seasonality of power load and the PACF plot results (figure 3.5). The lags eight days prior do not seem to be attributed much, hence could be left out of the model without considerable loss in performance. Moreover, since $\phi_{\boldsymbol{P_{d-2}}}$ is so much larger than all the other attributions, a model with only one day of lag would likely also not lose much performance.

5.1.2 Second generation models

The second generation of models, described in chapter 4.5.2, were given NWP input variables in addition to the historical power load. Three different second generation models were

tested: (1) A model with temperature input; (2) a model with all the meteorological variables; and (3) a model where the meteorological variables were selected based on the importance attributed by Deep SHAP for the second model (CNN-2-2). These attributions are shown in figure 5.4. The figure was produced in the same manner as for figure 5.3. Hence, remember that the figure is a measure of the average absolute importance of each variable.

The feature attributions made by Deep SHAP suggested that the most important variables for the 2019 forecasts were historical load, temperature, humidity and pressure. While the least important variables were wind speed and cloud cover. Interestingly, the model seemed to put a large emphasis on the pressure predictions in Fagernes ($\boldsymbol{p}_F$). Judging by the importance of the other pressure variables, it was deemed unlikely that the pressure of *one* location had a much greater predictive ability than the other locations. Moreover, by observing the seasonal performance of CNN-2-2 shown in table 5.3, there seemed to be something inherently wrong with the model. Based on these results, it was decided to omit cloud cover and wind speed from the third model of the second generation. Note that CNN-2-2 may not have had the appropriate architecture and hyper-parameters to facilitate better performance with cloud cover and wind speed. CNN-2-2 performed the best out of the three models during winter, which may indicate that these variables are more important in this time period. Intuitively, this makes sense since wind speed and temperature are connected to how cold the weather may "feel" during winter.

The improvement compared to the first generation (table 5.1) by adding temperature (CNN-2-1) can clearly be seen from table 5.3. However, as discussed above, simply adding more meteorological variables (CNN-2-2) actually worsened the performance for every season except winter. This was the motivation of omitting the least important variables (CNN-2-3), and resulted in the best performance of the three models. Likewise as the first generation results, the best model for each season and metric is in bold. It can be seen that there is improvement across the board compared to the first generation metrics. Particularly there is markedly better performance for winter, spring and autumn. The performance in summer however, did not seem to improve much with the addition of meteorological variables. This is in line with the characteristics of Norwegian consumption patterns discussed in chapter

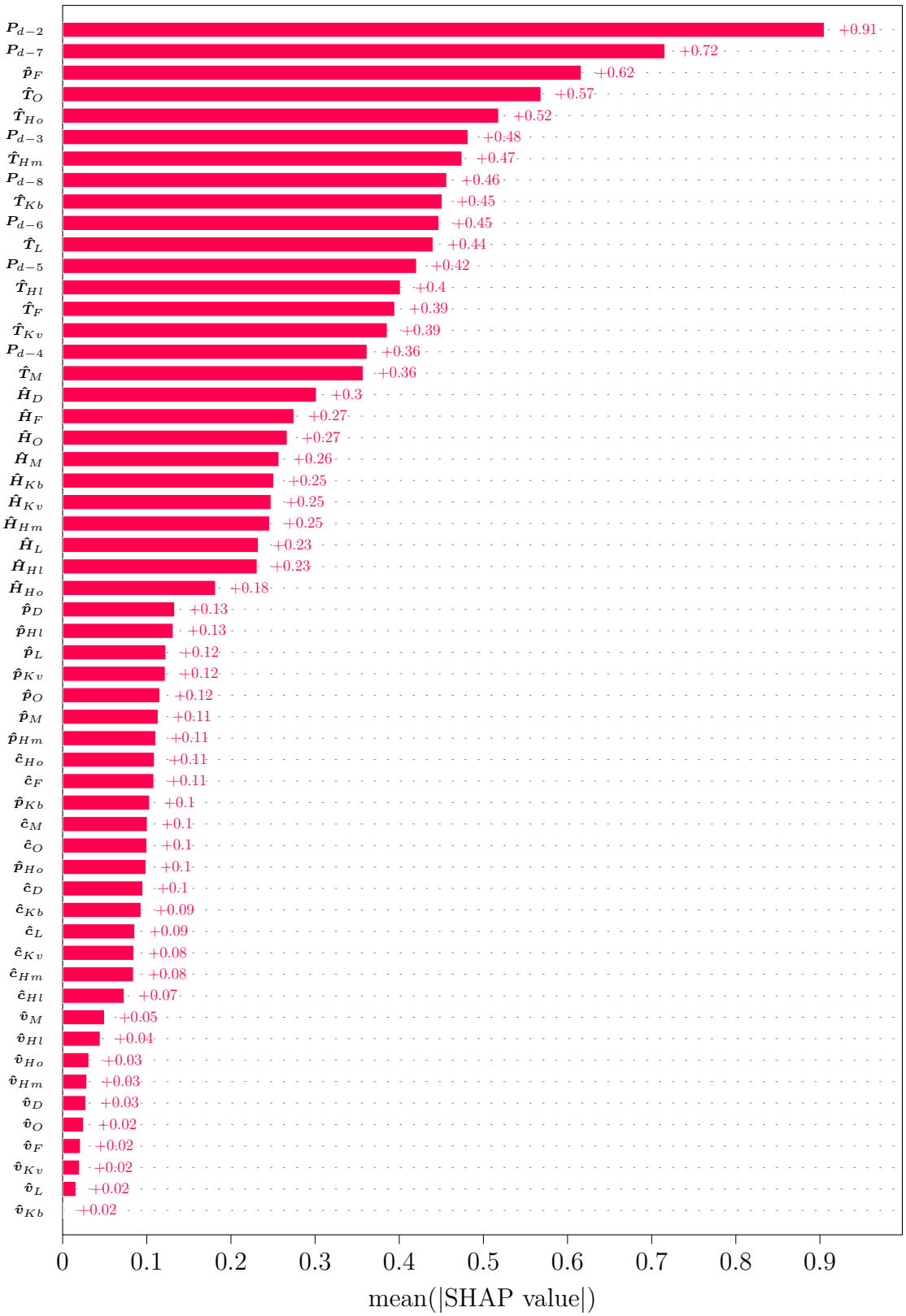


Figure 5.4: Global feature importance for CNN-2-2. Each SHAP value is the average of the average across output hours and instances.

Table 5.3: The seasonal metrics for the second generation models in MAPE/RMSE.

Model	Winter		Spring		Summer		Autumn		Full year	
CNN-2-1	3.43	244	4.50	222	**4.23**	**155**	3.13	171	3.82	201
CNN-2-2	**3.06**	225	4.99	244	5.11	179	3.11	168	4.06	206
CNN-2-3	3.13	**224**	**4.37**	**218**	4.30	161	**2.95**	**158**	**3.69**	**193**

Table 5.4: Metrics by day of the week for the second generation models in MAPE/RMSE.

Model	Monday		Tuesday		Wednesday		Thursday		Friday		Saturday		Sunday	
CNN-2-1	**3.73**	**196**	3.27	192	4.46	237	3.82	**203**	3.95	220	3.87	185	**3.77**	176
CNN-2-2	4.21	215	3.45	193	4.39	236	4.21	220	4.14	217	3.97	**174**	4.18	185
CNN-2-3	3.97	205	**2.89**	**180**	**4.05**	**216**	**3.75**	205	**3.52**	**190**	**3.83**	182	3.89	**174**

3.1.2. I.e., summer power load is not as sensitive to weather variables compared to the other seasons, hence the models did not see much improvement during this period. The performance of each models by day of the week can be seen from table 5.4. Compared to the first generation performances (table 5.2) all days saw an increase in performance across all models. Monday and Wednesday appear to have the worst performances, but the performances of all the days are more or less the same. That is, except for Tuesday which may have been the easiest to forecast.

Lastly, to evaluate the shape of the load forecasts, the first week of forecasts for 2019 is shown in figure 5.5. The forecasts shown are only the ones made by CNN-2-1, since the other two were comparable in shape and accuracy. Compared to the first generation results for the first week (figure 5.1), there seems to be better accuracy both in terms of absolute deviation, but also in the shape of the forecast load. The forecasts resemble the actual load profiles of each day much more closely than the first generation forecasts. Moreover, CNN-2-1 appeared to be able to somewhat predict the right level of load on New Year's Day which was not expected. Two additional months of the second generation forecasts were produced, but were left out for the sake of brevity. These can be found in appendix B.

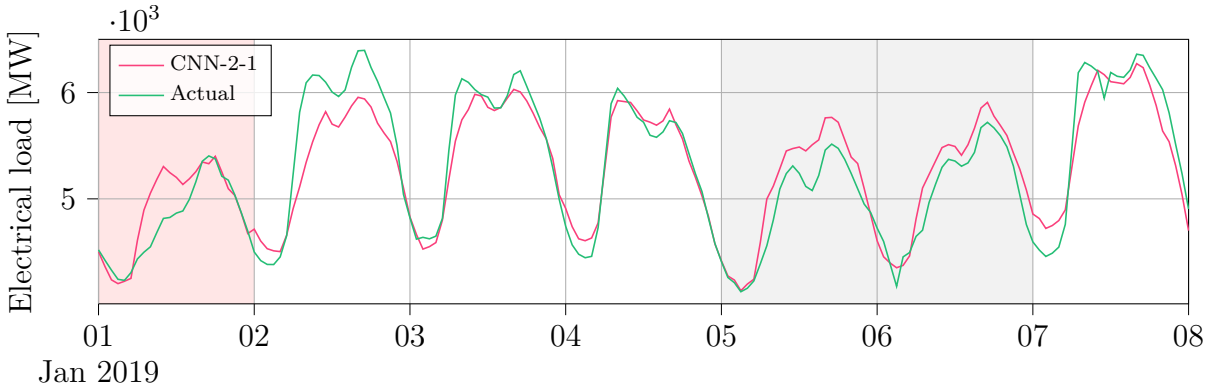


Figure 5.5: CNN-2-1 forecasting results for the first week of 2019.

Table 5.5: The seasonal metrics including Easter for the third generation models in MAPE/RMSE.

Model	Winter		Spring		Summer		Autumn		Easter		Full year	
CNN-3-1	3.07	218	**3.62**	218	**3.39**	**118**	2.92	**156**	6.54	233	3.25	**170**
CNN-3-2	**2.96**	**213**	3.64	**174**	3.46	123	**2.87**	159	**4.52**	**175**	**3.23**	171

5.1.3 Third generation models

The third generation models (see section 4.5.2 for details) were an extension of the best performing second generation model (CNN-2-3). They were given calendar features which indicate the season, month, day of the week and whether the forecast is on a public holiday or not. Refer to section 3.1.3 for an overview of the Norwegian public holidays and their effect on the power load.

Table 5.5 shows the seasonal performances of the two third generation models. The first model (CNN-3-1) had an increase in performance across all seasons and the entire year compared to the second generation (table 5.3). The forecasting result of CNN-3-1 for the first week of 2019 is shown in figure 5.6. Although the forecast of January 7th appears to have overshot slightly, the forecasts of the rest of the week are both smoother and more accurate than the second generation first week (figure 5.5).

Table 5.6 shows the performances of the third generation models by each day of the week. Compared to the second generation performances (table 5.4), all days saw an improvement. Mondays, Thursdays and Sundays saw the greatest increases. Surprisingly however, Wednes-

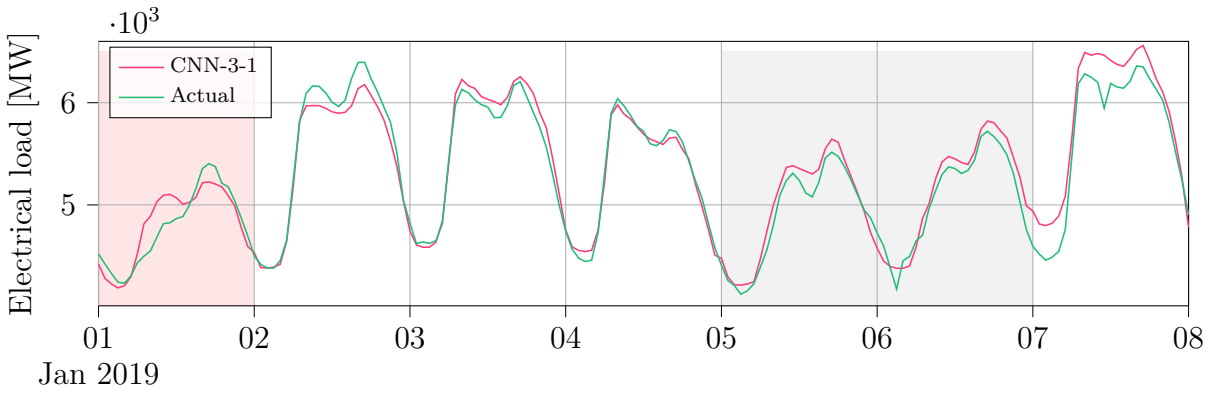


Figure 5.6: CNN-3-1 forecasting results for the first week of 2019.

Table 5.6: Metrics by day of the week for the third generation models in MAPE/RMSE.

Model	Monday		Tuesday		Wednesday		Thursday		Friday		Saturday		Sunday	
CNN-3-1	2.94	151	2.70	167	3.83	203	3.26	168	3.15	169	3.44	160	3.41	161
CNN-3-2	2.88	146	2.73	150	3.73	202	3.14	166	3.64	194	3.64	166	3.18	145

day, Saturdays and Sundays did not improve much by including the calendar features. The second generation and first generation appeared to have learned the weekly cycle without the use of calendar variables (see figures 5.1 and 5.5), but adding the calendar variables may have made the harder days easier to predict.

Regardless of the immediate improvements, unintended artifacts were observed on some days. This is shown in figure 5.7 with the artifacts in frame. Particularly, for the eight hour (07AM) on Palm Sunday and Easter Sunday, the forecast power load suddenly collapses before recovering again. This type of pattern, at the time of day when power load usually sees a peak (see section 3.1.2), was thought to be a bug and would have been questioned if the model was in deployment. Note from figure 5.7, the relatively consistent electrical load in the two weeks prior to the first artifact. Why were the forecasts for Palm Sunday and Easter Sunday any different from the Sunday prior to Palm Sunday? In other words, which changes in the model's inputs made such a large change in the model's output compared to the previous Sunday's forecast? Questions like these may be answered with an explanation as described in section 4.7.3, where a single forecast is explained with hand-picked reference(s) in mind. Hence, to get an idea of what was going on at these two hours, local explanations

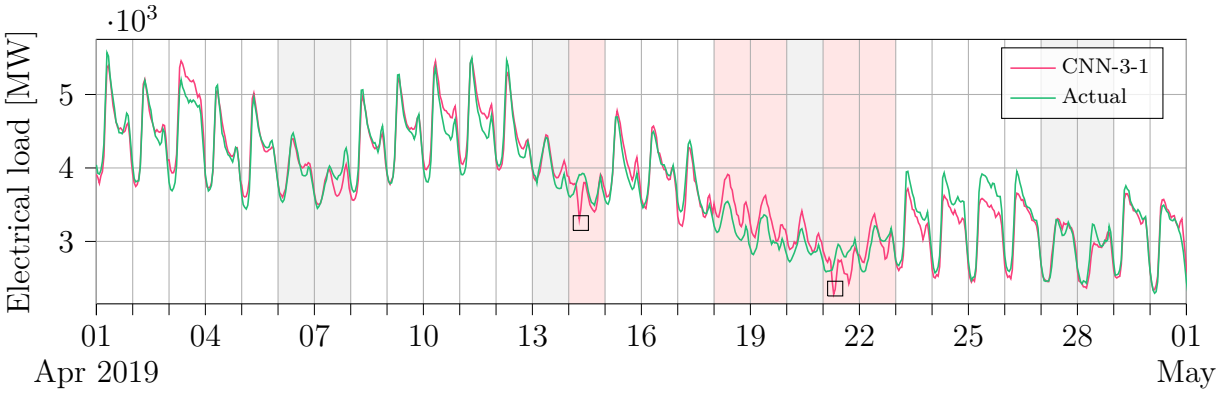
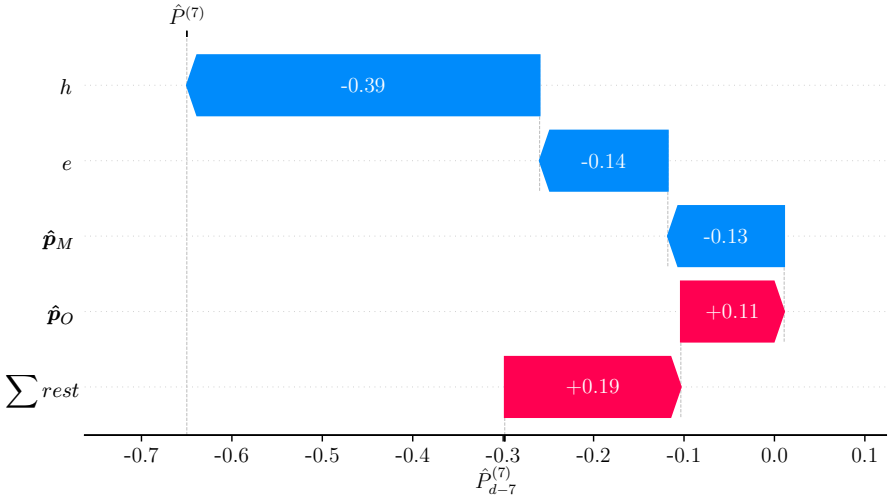


Figure 5.7: CNN-3-1 forecasts for April 2019. The boxed hours are unexpected behaviours of the model.

with Deep SHAP were made. The Sunday prior to Palm Sunday was chosen as the single reference sample. Keep in mind, that as a consequence of the choice of reference, model inputs of Palm Sunday and Easter Sunday which see no difference compared to the reference will be given little importance. Notably, calendar values like day of the week, season and month will and should by property 2 of SHAP give $\phi_i = 0$, since $\boldsymbol{d}^0 = \boldsymbol{d}$, $\boldsymbol{s}^0 = \boldsymbol{s}$ and $\boldsymbol{m}^0 = \boldsymbol{m}$.

The local explanation of the first artifact is shown in figure 5.8. The input values for the sample (x) and the reference ($E[x]$) of the top four features are also shown next to the explanation. SHAP attributions for all the continuous features were aggregated by use of equation 4.12 and 4.13. By not taking the absolute value of each SHAP value, the sum of the aggregated SHAP values and the calendar SHAP values should add up to the difference from the expected prediction (the reference) by property 1 of SHAP. I.e., $\sum \phi = \hat{P}^{(7)} - E[\hat{P}^{(7)}] = \hat{P}^{(7)} - \hat{P}^{(7)}_{d-7}$. This can be seen from figure 5.8, where each feature contributes to push the prediction up or down from the reference. Note again that the feature contributions of h and e are SHAP values of *one* feature; while the contributions of bold variables, such as the pressure in Moss, $\hat{\boldsymbol{p}}_M$, are aggregated contributions of *twenty-four* features (see section 4.7). Hence, the aggregated SHAP values may have terms which cancel each other out, even if they are very large. An explanation like this may therefore not be very well suited for feature importance.

From the explanation of the first artifact it appears that h being "ON" contributed to pushing

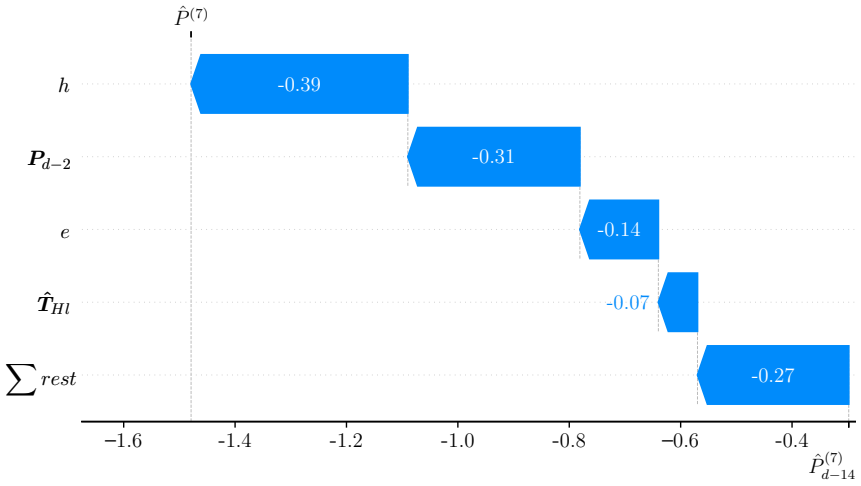


	x	$E[x]$
h	1	0
e	1	0
$\bar{\boldsymbol{p}}_M$	1.96	0.86
$\bar{\boldsymbol{p}}_O$	1.62	0.56

Figure 5.8: Local explanation of the forecast for April 14th 7AM (Palm Sunday).

the load by -0.39, while e contributed with -0.14. Public holidays are known to reduce load compared to normal (see section 3.1.3), but the negative contribution of h for Palm Sunday was more than intended when implementing calendar features. Meanwhile, the opposite contributions of $\hat{\boldsymbol{p}}_M$ and $\hat{\boldsymbol{p}}_O$ appear to cancel each other out, even though the daily average pressure, $\bar{\boldsymbol{p}}$, was greater than the reference pressure in both cities on Palm Sunday. The reason for this is unclear, but as discussed above, the features of other important continuous variables may have cancelled each other out.

The explanation for the second artifact is shown in figure 5.9. Likewise as the first artifact, the Sunday prior to Palm Sunday was used as reference to estimate the SHAP values. For the Easter Sunday 7AM forecast, h is also the most contributing feature for explaining the difference to reference. Once again, this feature pushed the load down by -0.39, which was more than intended. The second-most important feature for the Easter Sunday forecast was $\boldsymbol{P}_{d-2}$. As can be seen from the average values of the sample versus reference, the negative contribution makes sense. This can also be seen graphically from figure 5.7 since the load is trending downwards prior to Easter Sunday. It can also be seen that the sum of all the other feature contributions add up to -0.27, i.e., they push the forecast of Easter Sunday down compared to the reference. These contributions are due to differences in weather of different locations and differences in observed power load lags. The contribution of $\hat{\boldsymbol{T}}_{Hl}$ and its two average values hint that it may be due to an increase in temperature. Note again, however,



	x	$E[x]$
h	1	0
$\bar{\boldsymbol{P}}_{d-2}$	-0.67	0.23
e	1	0
$\bar{\boldsymbol{T}}_{Hl}$	0.25	-0.06

Figure 5.9: Local explanation of the forecast for April 21st 7AM (Easter Sunday).

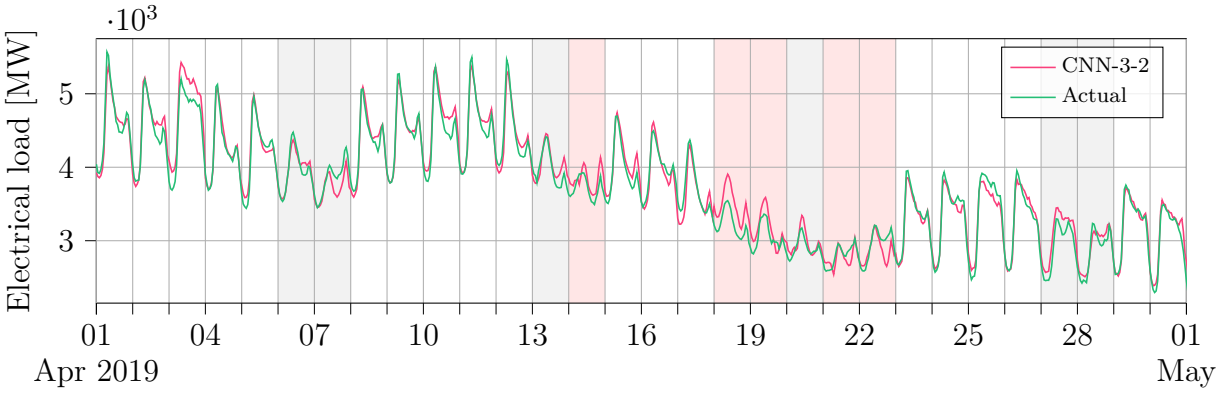


Figure 5.10: CNN-3-2 forecasts for April 2019.

that they are *not* due to any other calendar features because of the choice of reference.

Based on the explanations of the two artifacts it was decided that the implementation of the holiday feature had to change. Moreover, since the Easter feature covered the holiday effect for Easter forecasts sufficiently, all the Easter holidays were set to $h = 0$. Additionally, later evidence indicated that the abnormal behaviour of the holiday feature also appeared on forecasts where $d_6 = 1$ and $h = 1$ coincided. This may have been due to the fact that the load profiles of some holidays resemble those seen during weekends; hence, the presence of both d_6 and h may cause the model to reduce the load twice.

The second model of the third generation (CNN-3-2) was therefore implemented with $h = 1$ on all public holidays, except during Easter and when $d_6 = 1$. From table 5.5 it can be seen

that the change in implementation for CNN-3-2 increased performance during 2019 Easter by 30.9%. These improvements can also be seen visually from figure 5.10, which shows the CNN-3-2 forecasts of April 2019. The artifacts are almost gone, but not entirely, which means that the presence of the holiday feature may not have been the only problem. Also, acute performance improvements to the development year do not necessarily carry over to unseen data, hence it is important to not "unconsciously" bias towards 2019.

5.2 Model evaluation

The second part of results from the thesis were the ones made during the deployment of the last model (CNN-3-2). The methodology is described in detail in section 4.6. These results can be separated as: (1) Results of the expanding window test and the held out year (2020); and (2) selected local explanations from 2020 using Deep SHAP.

5.2.1 Expanding window test

The seasonal results from the expanding window test (see section 4.6.1 for details) are shown in table 5.7. The results suggest that the development of an electrical load model for NO1 was successful. The model appears to be able to generalize across multiple years and, surprisingly, performed the best during the hold-out year. Also, based on the forecasting accuracy for 2017 it appears that the model is able to sufficiently learn the electrical load patterns from only three years of training data. This can be seen from the fact that the 2017 full year accuracy is the second-best of all years. This may mean that there is an opportunity to further increase model complexity without over-fitting. Moreover, the full forecast results (see appendix C) appear very accurate most of the year. There are some situations which demand further development if the model was to be put in official deployment, but for the purposes of the thesis, the results are satisfactory.

Forecasting accuracy by model output hours in MAPE is given in figure 5.11. Note that by definition of the midnight termin time (see section 4.2) and the day-ahead horizon, the output of the model by zero-index corresponds to the actual clock hour of the forecast day. I.e., the 0th model output corresponds to 12PM, the 1st output to 1AM, and so on. As can

Table 5.7: The seasonal metrics of the expanding window test in MAPE/RMSE.

Test year	Winter		Spring		Summer		Autumn		Full year	
2017	2.52	186	3.71	193	2.96	99	2.73	152	2.99	161
2018	3.07	246	3.27	172	2.89	93	2.78	150	3.01	170
2019	2.96	213	3.64	174	3.46	123	2.87	159	3.23	171
2020	2.92	157	3.26	165	2.66	94	2.70	151	2.88	157
Average	2.87	201	3.47	176	2.99	102	2.77	153	3.03	165

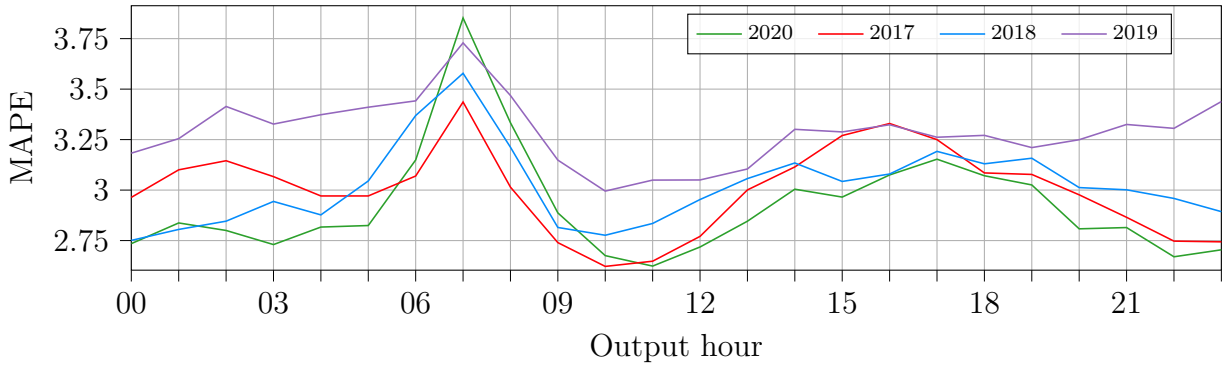


Figure 5.11: Expanding window MAPEs by model output hour.

be seen, 7AM appears to be the hardest hour to forecast across all years. Moreover, the curves of the forecasting accuracy seem to resemble that of the daily power load curve (see section 3.1.2). This suggests that the hours with the largest power load are also the hardest to forecast. 2019 was the worst year and, in contrast to the other years, the model seemed to have struggled with the hours closer to midnight. The rest of the years look comparable with respect to performance, which further substantiates the observation made above; the fact that the model is able to learn with as little as three years of training data.

5.2.2 Interpretation of selected forecasts

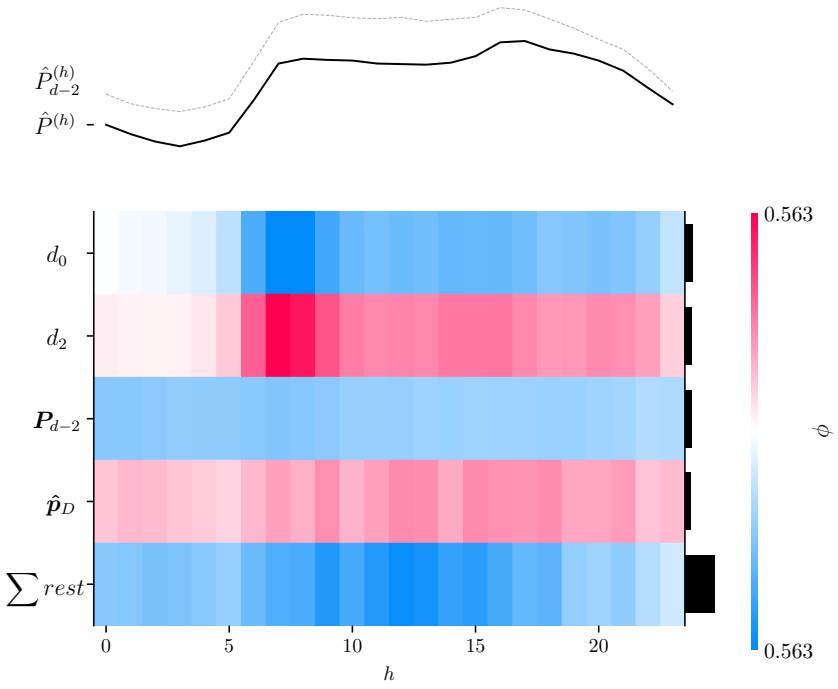
From the forecasts of power load in 2020 (appendix C), a few select days were analysed with local explanations using Deep SHAP (see section 4.7.3). Conspicuous or very wrong forecasts were selected in order to find the underlying cause. Moreover, every selected sample presented an opportunity to experiment with different background data assumptions. As described in section 4.7.1, the choice of appropriate background data is essential to derive meaningful

results from Deep SHAP, but is as of yet an open-ended question (Sturmfels et al., 2020).

The first selected sample was the forecast made for Wednesday, January 8th 2020. This day was chosen because both the actual and forecast power load saw a significant drop across the day compared to the rest of the week. As known from the similarity of the weekday load profiles (see section 3.1.2), the level of electrical load usually remains about the same for all weekdays; hence, it was expected that the forecast load for Wednesday would remain at about the same level as the Tuesday and Monday. The fact that the model was able to correctly predict the drop in load, motivates the need for an explanation of how it was able to do so. In other words, what made the forecast for Wednesday any different from the forecast of, say, Monday?

In order to answer this question, a local explanation was made for January 8th using the data from Monday as a reference. Note that by only using *one* background sample, the explanations made by Deep SHAP are essentially DeepLIFT explanations (this is explained in detail in section 2.4.3). Moreover, SHAP explanations are particular to each output hour, hence *one* sample will produce 24 explanations. Local explanations for one output hour were illustrated and explained for figures 5.8 and 5.9. By stacking 24 of these explanations together, one for every output hour, it is possible to get an explanation of the entire day as a function of the hour h. This is shown in figure 5.12 for the forecast of January 8th as a heatmap of the SHAP values for each particular hour and feature. The forecast values of January 8th ($\hat{P}^{(h)}$) and the reference ($\hat{P}^{(h)}_{d-2}$) are also shown for each hour. The top four features were sorted by absolute average contribution across all hours.

From the explanation of January 8th it can be seen that d_0 and d_2 ranked as the most contributing features. However, the two calendar features also perfectly cancel each other out. The fact that the sample is not a Monday ($d_0 = 0$), but a Wednesday ($d_2 = 1$), and that the reference is the opposite appears to even out the contributions. Based on the weekly load profiles (figure 3.4), this is intuitive since weekdays are expected to be about the same. Interestingly, Deep SHAP attributes these calendar features the most importance during the morning peak, but no importance during the night. This may be due to there being less variation in load during the night and more variation during the day from subtle differences



	x	$E[x]$
d_0	0	1
d_2	1	0
$\bar{\boldsymbol{P}}_{d-2}$	0.95	0.82
$\bar{\boldsymbol{p}}_D$	0.24	0.76

Figure 5.12: Local explanation of the forecast for January 8th 2020 as a function of the output hour. As background data, a single reference sample from January 6th 2020 was used.

in human consumption patterns.

The calendar features do not explain the drop in load, however. The fact that the average of the observed $d-2$ load lags was slightly lower than the reference ($\bar{\boldsymbol{P}}_{d-2} = 0.95$ versus $E[\bar{\boldsymbol{P}}_{d-2}] = 0.82$) contributed to pushing the forecast down. This contradicts the effect one would expect from an increase in observed load, i.e., that increase leads to increased forecast. It may be that there are unknown feature interactions due to the feature independence assumption (see section 2.4.3). In fact, it appears that the sum of all the other features ($\sum rest$) add up to why the forecast was lower than expected.

To further investigate the other features, an explanation with all the features was made, and it appeared that the temperature differences in the different locations all contributed to push the load down slightly. Hence, it seems that the main contributor of the forecast was the fact that the temperature predictions were larger for January 8th than for the reference in all ten locations. The Oslo temperature predictions in degrees Celsius as a function of the hour of the day is shown in figure 5.13. The two curves are the predictions for January

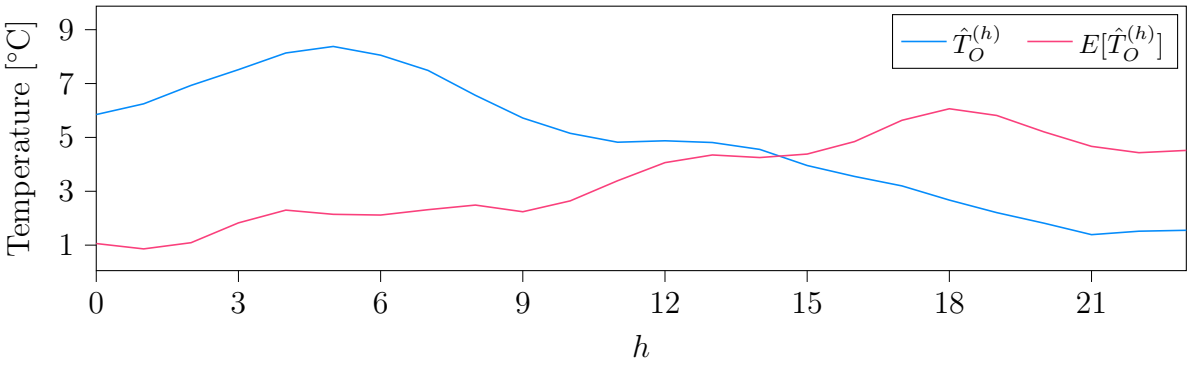


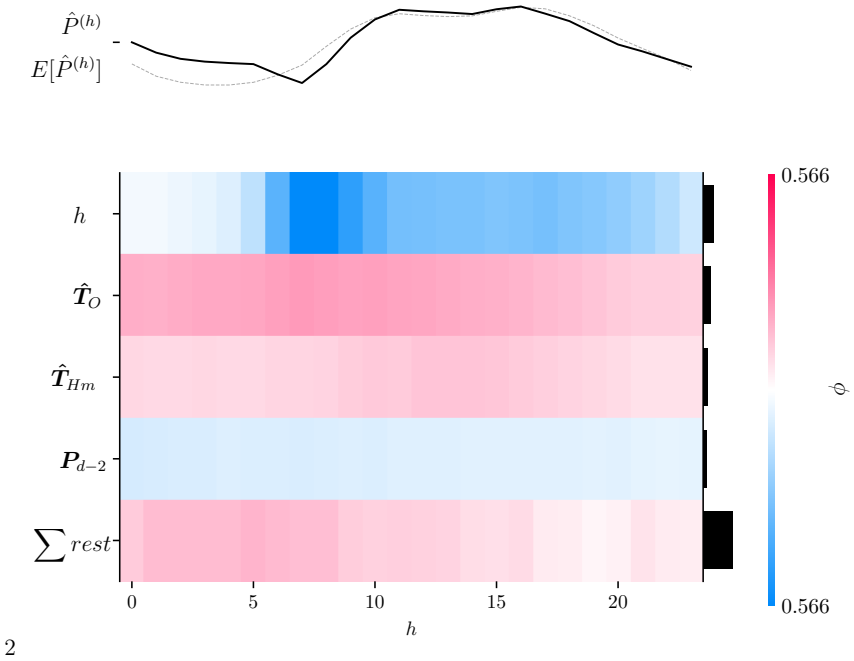
Figure 5.13: Temperature forecasts in Oslo for January 8th ($\hat{T}_O^{(h)}$) and January 6th ($E[\hat{T}_O^{(h)}]$).

8th ($\hat{T}_O^{(h)}$) and the reference ($E[\hat{T}_O^{(h)}]$). It can be seen that the temperature is higher (which will drive load down) for the selected sample, but decreases throughout the day. Similar observations can be made for the other locations, but are left out for brevity's sake.

The second selected sample was the forecast for Saturday, December 26th 2020. This day was chosen because another artifact, like the ones from section 5.1.3, was observed. A Deep SHAP explanation for this forecast is shown in figure 5.14. The heatmap was produced in the same way as figure 5.12, but instead of using a single reference, the three Saturdays prior to the sample were chosen as background data. Hence, $E[\hat{P}^{(h)}]$ is an average over these forecasts. The reasoning for this choice was that these forecasts looked representative of what one would expect if the sample took place on a regular Saturday.

As seen from the explanation, the holiday feature was the problem once again. Particularly, the presence of h appears to push down the load for most of the day, but considerably during the morning hours. This effect was unintended and would have to be dealt with in a similar way as was done for the third generation models. It seems that indicating the presence of a holiday during the weekend may be unnecessary, and that it may be better to model public holidays as Saturdays or Sundays. This can be seen from the fact that if $h = 0$, then the forecast would have been slightly higher than expected forecast and would in fact have been more accurate (as seen from the actual observed electrical load in appendix C).

The third chosen sample was the forecast made for Tuesday, May 12th 2020. This forecast was chosen because the model suddenly changes load profile after three/four weeks of con-



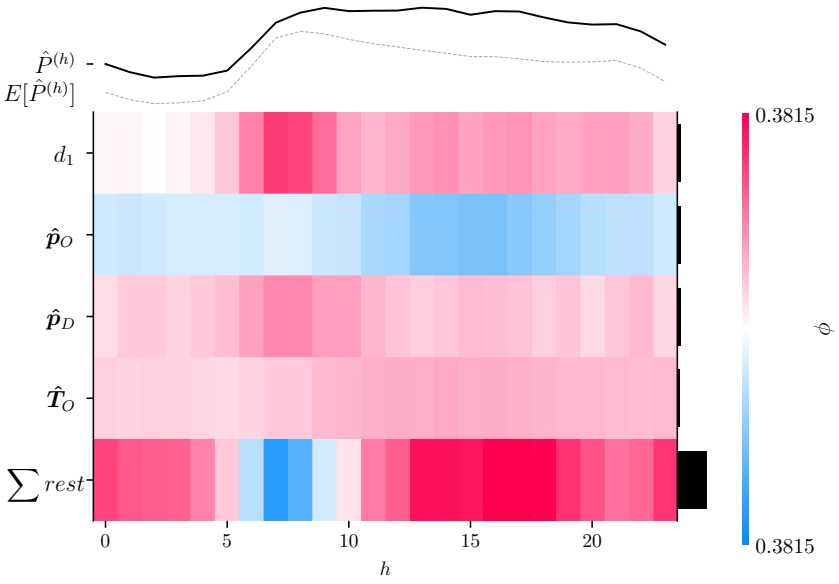
	x	$E[x]$
h	1	0
$\bar{\boldsymbol{T}}_O$	-0.77	-0.39
$\bar{\boldsymbol{T}}_{Hm}$	-1.09	-0.50
$\bar{\boldsymbol{P}}_{d-2}$	1.01	1.22

Figure 5.14: Local explanation of the forecast for December 26th 2020 as a function of the output hour. The background data were the three prior Saturdays.

sistent weekday load profiles. I.e., instead of having a sharp peak in the morning, before decreasing towards the evening, the forecast is more steady across the day. And although the forecast overshoots the actual observed values, the model managed to predict the change in load profile. The explanation for this forecast is shown in figure 5.15. The background data for this explanation were the three previous weeks of weekdays prior to May 12th.

As seen from the figure, the largest contribution seems to be attributed to $d_1 = 1$ which increased the load across the day compared to the background samples. However, notice that the morning hours contribution of $\sum rest$ cancels out d_1 during these hours. Upon further inspection it appears that this, again, are the sum of contributions from the other day of the week variables. I.e., it seems that the model puts little difference in prediction due to it being a Tuesday. This is also consistent with the observations made in regards to weekday similarity in section 3.1.2.

While the contributions of $\hat{\boldsymbol{p}}_O$ and $\hat{\boldsymbol{p}}_D$ are larger than $\hat{\boldsymbol{T}}_O$, inspection of the rest of the features suggested that the difference in predictions may have been due to differences in temperature



	x	$E[x]$
d_1	1	0.2
$\bar{\boldsymbol{p}}_O$	0.09	0.63
$\bar{\boldsymbol{p}}_D$	0.39	0.92
$\bar{\boldsymbol{T}}_O$	-0.37	0.28

Figure 5.15: Local explanation of the forecast for May 12th 2020 as a function of the output hour. The background data were the three past weeks of weekday forecasts.

predictions. Particularly, it was found that out of the top ten features, four of them were temperature variables ($\hat{\boldsymbol{T}}_O$, $\hat{\boldsymbol{T}}_{Hl}$, $\hat{\boldsymbol{T}}_L$ and $\hat{\boldsymbol{T}}_{Kv}$). All of these had large positive SHAP values across all hours, and the temperature forecasts for these locations were much lower for May 12th than the averages over the background samples. This is illustrated for Oslo in particular in figure 5.16. Notice how the greatest differences in temperature predictions are observed in the range of about $h \in [11, 18]$. This also coincides well with where the $\sum rest$ contributions are the largest (most vibrant red). As known, electrical load is particularly sensitive to temperature changes (see e.g., the MI matrix from table 3.8). Hence, the differences in temperature predictions seem to have been the main drivers to push the forecast load above the expected load profile during the day. Note however, that although the model got the shape of the forecast right, it missed on the level of load.

5.3 Discussion of the results

Based on the performance on unseen data (2020), the expanding window results (see table 5.7) and visual confirmation of the 2020 forecasts (see appendix C), the development of a

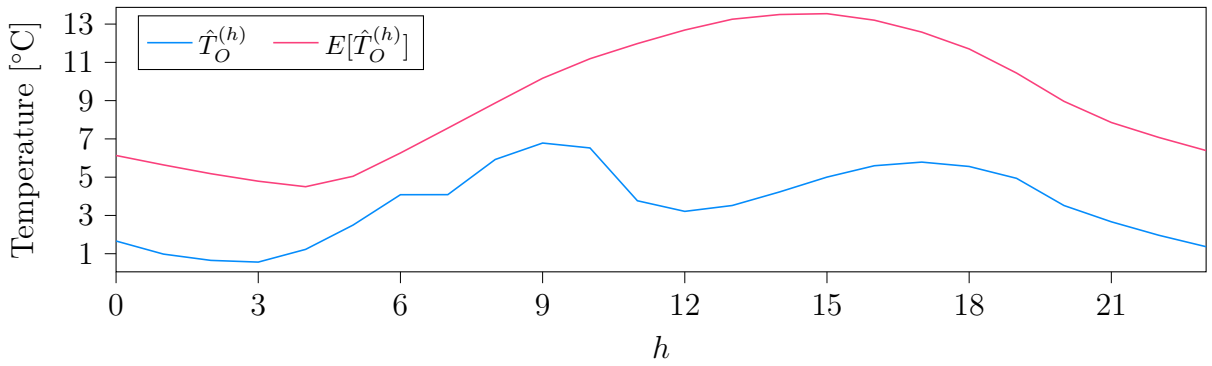


Figure 5.16: Temperature forecast in Oslo for May 12th and the expected temperature forecast.

load forecasting model for NO1 was a success. Moreover, the use of convolutional layers to extract weather features from a spatial grid (see equation 4.7) appears a promising area for future research. Particularly, instead of limiting the number of locations to ten, one could drastically increase the number of grid points for densely populated areas (such as Oslo and surrounding areas), and slightly increase the number of grid points for locations not represented in the thesis model. This may lead to over-fitting issues due to the number of parameters (one kernel for each location), but with the use of convolutional layers it may take a very large number of grid points before this occurs.

No forecasting model is without flaws however, and neither is the one developed in this thesis. As seen from the selected samples during model deployment, the model made some unacceptable mistakes which would have to be alleviated if it was to be put into production. When a black-box model makes a poor forecast, trust in the model's capabilities may be lost. If the model messed up once, how can you be sure it will not happen again? With the use of XAI, however, it is possible peek into the black box; find out what might be at fault; and make adjustments to the model. Alternatively, if the underlying causes of the model error are identified, then similar scenarios in the future are scrutinized. For instance, as seen from the explanation of the second selected sample (figure 5.14), the model appears to act out when forecasts are on a Saturday public holiday. Likewise, during model development, it was found that the presence of a holiday on a Sunday caused the same error (see figure 5.8 and 5.9). There are then two choices: to change the implementation of the holiday variable;

or to take every future forecast made for a Saturday holiday with a grain of salt.

Another insight from the Deep SHAP results was that the pressure predictions may have been given too much importance (see figure 5.4, 5.8 and 5.15) because of neural network architecture design choices (see appendix B). Particularly, the number of extracted pressure features for the last model in development (CNN-3-2) was 240, while the number of load and temperature features were 96 and 168, respectively. It was therefore suspected that Deep SHAP may have given pressure more importance due to the larger number of activation differences caused by differences in pressure predictions. On the other hand, the model may in fact be emphasizing pressure, and Deep SHAP may then, in turn, have given accurate attributions for pressure. It is difficult to know which one of the above, model or Deep SHAP, is at fault because there is no ground truth explanation for a black box model (Sturmfels et al., 2020). However, the acquired insights from using Deep SHAP suggest that improvements to the model can be made. If, in turn, an increase in performance or model robustness would be achieved from these findings, then Deep SHAP will have served its purpose in shedding some light on the forecasting model. Hence, it may not matter what is the ground truth explanation, as long as the given explanation leads to some form of increased understanding of the model.

Model developers and domain experts are the most frequent audience for XAI in deployment (Bhatt et al., 2020), but may not be the most important. Hong (2014) noted in his doctoral thesis that shareholders require full transparency of production models in high-stake situations, but Bhatt et al. (2020) found that in practice, stakeholders and decision makers are not kept in-the-loop when it comes to XAI. Therefore, in order to facilitate increased use of XAI for a non-expert audience, the use of intuitive background data selection is crucial. This is especially true for time series forecasting problems, where it is natural to look in retrospect when faced with something unexpected. For instance, when explaining a forecast made for tomorrow by some black-box model, it is in the line of human reasoning to ask the question: "what is different from yesterday's forecast?". In the specific case of load forecasting (as discussed in detail in section 4.7.1) and its weekly periodicity, another form of human-centric question is to ask: "if electrical load repeats itself, why is the forecast any

different from the previous period?". These sort of questions were investigated for a model in deployment using Deep SHAP (see figures 5.12, 5.14 and 5.15), and the resulting explanations also gave accessible answers. For example, the explanation for the increased load on May 12th appeared to have mainly been due to differences in temperature predictions from the background data. Roughly speaking, put into words one could say something like: "The forecast was higher than the previous three weeks mostly due to decreased temperature forecasts over multiple locations". This is the sort of explanation anyone could understand, and can be used in conjunction to the SHAP visualizations.

While the choice of similar background samples is intuitive, it has some drawbacks. For starters, such explanations do not provide the full picture of what lead to the forecast. Particularly, calendar variables and other variables with similar input values to the background samples will be ignored due to zero difference from reference. Hence, if a forecast on a Saturday in March is compared to previous Saturdays in March, then the day of the week, season, and month of the year variables will be attributed zero importance. This may lead someone to wrongly think that these are not important. However, when using a difference from reference philosophy (see section 2.4.1), it is important to keep in mind that these may still have been important to lead up to the baseline; they were just not what made up the difference to what was expected from the background samples. Another drawback is that the mathematical rigorousness of these assumptions with respect to the SHAP properties is unknown. I.e., since SHAP relies on estimating SHAP values by sampling over a, preferably, *large* number of background samples, it is unclear how choosing a *small* number of samples affects the estimation accuracy. However, choosing a small number (or just one) of samples is more in line with the original DeepLIFT (see section 2.4.1) method, and is not necessarily wrong. As stated previously, as long as an explanation leads to increased understanding or model performance in some way, then the explanation has served its purpose.

Chapter 6

Conclusion

The three specific contributions of the thesis was to (1) develop an electrical load forecasting model for the Norwegian price region, NO1; (2) use XAI, in this case Deep SHAP, during development to improve the model performance; and (3) interpret forecasts in deployment by use of an XAI tool. All of these three were carried out and the results were presented in chapter 5. However, the more important contribution of the thesis was to serve as as a first work in the use of XAI for Norwegian load forecasts and to provide directions for future research. The thesis has demonstrated how XAI, particularly Deep SHAP, can be used to improve and interpret a black-box ML load forecasting model. Moreover, domain knowledge of electrical load was leveraged in choice of SHAP background data to produce more intuitive explanations which are accessible to both practitioners and non-experts. Particularly, it was found that while global explanations may be well served sampling from the entire expected distribution of the load; local explanations may be better off by choosing background samples based on the periodic nature of electrical load.

Regardless of the intuitive soundness of these local background choices, more research is needed with respect to choosing background data for SHAP value estimation. Particularly, such choices should derive from close cooperation with the intended audience. I.e., if the targeted audience are, say, balancing operators or domain experts then the explanations should be tailored to their needs. Conversely, if the explanation is targeted towards shareholders

or decision-makers, they may require different background assumptions. Moreover, little is known in how the background assumptions affect the mathematical rigorousness of SHAP, which may prove to violate the properties of the method. However, it is as of now unclear whether this is an issue or not since such explanations still give increased understanding of the model.

New and smarter technology is part of the solution towards total decarbonization of the power system. Complex black-box AI models offer lower power system losses through more accurate load forecasts, which in turn lower emissions. With the dynamics of the power system and consumer landscapes changing, and more complex black-box models, like deep neural networks or large ensemble models, are put into deployment, XAI methods are needed in order to facilitate transparency, trust and increased use of these forecasting models going forward. However, to trust the explanations, first one has to trust the framework behind the explanations. Hence, with more research in the mentioned areas, XAI methods have the potential of serving as crucial, trustworthy interlinks between the targeted audience of load forecasts and the models behind the forecasts.

Bibliography

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Available at: tensorflow.org.

Apadula, F., Bassini, A., Elli, A., and Scapin, S. (2012). Relationships between meteorological variables and monthly electricity demand. *Applied Energy*, 98:346356.

Arjunan, P., Poolla, K., and Miller, C. (2020). Energystar++: Towards more accurate and explanatory building energy benchmarking. *Applied Energy*, 276:115413.

Arrieta, A. B., Díaz-Rodríguez, N., Del Ser, J., Bennetot, A., Tabik, S., Barbado, A., García, S., Gil-López, S., Molina, D., Benjamins, R., et al. (2020). Explainable artificial intelligence (xai): Concepts, taxonomies, opportunities and challenges toward responsible ai. *Information Fusion*, 58:82–115.

Bailey, G. (2013). CS1114: Convolution. Department of Computer Science at Cornell University.

Baliyan, A., Gaurav, K., and Mishra, S. K. (2015). A review of short term load forecasting using artificial neural network models. *Procedia Computer Science*, 48:121–125. International Conference on Computer, Communication and Convergence (ICCC 2015).

Ben Taieb, S., Bontempi, G., Atiya, A. F., and Sorjamaa, A. (2012). A review and comparison of strategies for multi-step ahead time series forecasting based on the nn5 forecasting competition. *Expert Systems with Applications*, 39(8):7067 – 7083.

Bergstra, J. and Bengio, Y. (2012). Random search for hyper-parameter optimization. *The Journal of Machine Learning Research*, 13:281–305.

Bhatt, U., Xiang, A., Sharma, S., Weller, A., Taly, A., Jia, Y., Ghosh, J., Puri, R., Moura, J. M., and Eckersley, P. (2020). Explainable machine learning in deployment. In *Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency*, pages 648–657.

Bianchi, F. M., Maiorino, E., Kampffmeyer, M. C., Rizzi, A., and Jenssen, R. (2017). Recurrent neural networks for short-term load forecasting. *SpringerBriefs in Computer Science*.

Bishop, C. M. (2008). *Pattern Recognition and Machine Learning*. Springer.

Bolstad, D. A. (2020). Interpretation of electrical load forecasts using explainable artificial intelligence. Project thesis.

Brownlee, J. (2018). *Deep Learning for Time Series Forecasting: Predict the Future with MLPs, CNNs and LSTMs in Python*.

Chen, K., He, Z., Chen, K., Hu, J., and He, J. (2017). Solar energy forecasting with numerical weather predictions on a grid and convolutional networks. In *2017 IEEE Conference on Energy Internet and Energy System Integration (EI2)*, pages 1–5.

Chollet, F. et al. (2015). Keras. Available at: https://keras.io.

Claveria, O., Monte, E., and Torra, S. (2017). Data pre-processing for neural network-based forecasting: does it really matter? *Technological and Economic Development of Economy*, 23(5):709–725.

Clevert, D.-A., Unterthiner, T., and Hochreiter, S. (2016). Fast and accurate deep network learning by exponential linear units (elus).

Debnath, K. B. and Mourshed, M. (2018). Forecasting methods in energy planning models. *Renewable and Sustainable Energy Reviews*, 88:297–325.

Doshi-Velez, F., Kortz, M., Budish, R., Bavitz, C., Gershman, S., O'Brien, D., Scott, K., Schieber, S., Waldo, J., Weinberger, D., et al. (2017). Accountability of ai under the law: The role of explanation. *arXiv preprint arXiv:1711.01134*.

Dumoulin, V. and Visin, F. (2016). A guide to convolution arithmetic for deep learning.

Elvia AS (2020). Kraftsystemutredning 2020-2040: Hovedrapport Oslo, Akershus og Østfold.

Frogner, I.-L., Singleton, A., Køltzow, M., and Andrae, U. (2019). Convection-permitting ensembles: Challenges related to their design and use. *Quarterly Journal of the Royal Meteorological Society*, 145.

Gandhi, O., Rodríguez-Gallegos, C. D., and Srinivasan, D. (2016). Review of optimization of power dispatch in renewable energy system. In *2016 IEEE Innovative Smart Grid Technologies - Asia (ISGT-Asia)*, pages 250–257.

Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In Teh, Y. W. and Titterington, M., editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy. PMLR.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. Available at: http://www.deeplearningbook.org.

Grimaldo, A. I. and Novak, J. (2020). Combining machine learning with visual analytics for explainable forecasting of energy demand in prosumer scenarios. *Procedia Computer Science*, 175:525–532.

Grosse, R. (2018). CSC 321 Lecture 5: Multilayer perceptrons. Department of Computer Science of the University of Toronto.

Gu, J., Wang, Z., Kuen, J., Ma, L., Shahroudy, A., Shuai, B., Liu, T., Wang, X., Wang, G., Cai, J., and Chen, T. (2018). Recent advances in convolutional neural networks. *Pattern Recognition*, 77:354 – 377.

Hastie, T., Tibshirani, R., and Friedman, J. (2008). *The Elements of Statistical Learning*. Springer.

He, W. (2017). Load forecasting via deep neural networks. *Procedia Computer Science*, 122:308–314. 5th International Conference on Information Technology and Quantitative Management, ITQM 2017.

Higashiyama, K., Fujimoto, Y., and Hayashi, Y. (2018). Feature extraction of nwp data for wind power forecasting using 3d-convolutional neural networks. *Energy Procedia*, 155:350–358. 12th International Renewable Energy Storage Conference, IRES 2018, 13-15 March 2018, Düsseldorf, Germany.

Hong, T. (2010). Short term electric load forecasting.

Hong, T. (2014). Energy forecasting: past, present and future. *Foresight: The International Journal of Applied Forecasting*, 32:43–48.

Hong, T., Wang, P., and White, L. (2015). Weather station selection for electric load forecasting. *International Journal of Forecasting*, 31(2):286 – 295.

Horne, H., Roos, A., Magnussen, I. H., Buvik, M., and Langseth, B. (2020). Norge har et betydelig potensial for forbrukerfleksibilitet i sektorene bygg, transport og industri. *NVE*, Nr. 7.

Hyndman, R. and Athanasopoulos, G. (2018). Forecasting: principles and practice, 2nd edition. Accessed: 20/11/2020.

Ilic, I., Görgülü, B., Cevik, M., Gök, M., and Baydogan, M, G. (2020). Explainable boosted linear regression for time series forecasting.

James, G., Witten, D., Hastie, T., and Tibshirani, R. (2013). *An Introduction to Statistical Learning*. Springer.

Kuzlu, M., Cali, U., Sharma, V., and Güler, Ö. (2020). Gaining insight into solar photovoltaic power generation forecasting utilizing explainable artificial intelligence tools. *IEEE Access*, 8:187814–187823.

Lee, Y, G., Oh, J, Y., and Kim, G. (2020). Interpretation of load forecasting using explainable artificial intelligence techniques. *Transactions of the Korean Institute of Electrical Engineers*, 69(3):480–485.

Linnainmaa, S. (1970). The representation of the cumulative rounding error of an algorithm as a taylor expansion of the local rounding errors. Masters thesis.

Lundberg, S. M., Erion, G., Chen, H., DeGrave, A., Prutkin, J. M., Nair, B., Katz, R., Himmelfarb, J., Bansal, N., and Lee, S.-I. (2020). From local explanations to global understanding with explainable ai for trees. *Nature Machine Intelligence*, 2(1):2522–5839.

Lundberg, S. M. and Lee, S.-I. (2017). A unified approach to interpreting model predictions. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 30, pages 4765–4774. Curran Associates, Inc.

MacKay, D. J. C. (2003). *Information Theory, Inference and Learning Algorithms*. Cambridge University Press.

Makridakis, S., Spiliotis, E., and Assimakopoulos, V. (2018). Statistical and machine learning forecasting methods: Concerns and ways forward. *PLOS ONE*, 13(3):1–26.

McKinney, W. (2010). Data Structures for Statistical Computing in Python. In *Proceedings of the 9th Python in Science Conference*, pages 56 – 61.

Montgomery, D. C., Jennings, C. L., and Kulahci, M. (2008). *Introduction to Time Series Analysis and Forecasting*. John Wiley & Sons. Inc.

Nielsen, M. A. (2015). *Neural Networks and Deep Learning*. Determination Press.

Nord Pool (2021). Historical market data. Available at: https://nordpoolgroup.com.

NVE (2020). Kraftproduksjon. Available at: https://nve.no.

O'Malley, T., Bursztein, E., Long, J., Chollet, F., Jin, H., Invernizzi, L., et al. (2019). Keras Tuner. https://github.com/keras-team/keras-tuner.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Petrican, T., Vesa, A., Antal, M., Antal, C., Cioara, T., Anghel, I., and Salomie, I. (2018). Evaluating forecasting techniques for integrating household energy prosumers into smart grids. *2018 IEEE 14th International Conference on Intelligent Computer Communication and Processing*, pages 79–85.

Presthus, G. S. (2018). Developing and deploying machine learning models for online load forecasts. Available at: https://datascience.statnett.no.

Ruder, S. (2016). An overview of gradient descent optimization algorithms.

Shrikumar, A., Greenside, P., and Kundaje, A. (2019). Learning important features through propagating activation differences.

Spilde, D., Hodge, L. E., Magnussen, I. H., Hole, J., Buvik, M., and Horne, H. (2019). Strømforbruk mot 2040. *NVE*.

Statnett (2021a). Power system data. Available at: https://www.statnett.no.

Statnett (2021b). Reserve markets. Available at: https://www.statnett.no.

Sturmfels, P., Lundberg, S., and Lee, S.-I. (2020). Visualizing the impact of feature attribution baselines. *Distill*. https://distill.pub/2020/attribution-baselines.

Tian, C., Ma, J., Zhang, C., and Zhan, P. (2018). A deep neural network model for short-term load forecast based on long short-term memory network and convolutional neural network. *Energies*, 11.

Turek, M. (2018). Explainable artificial intelligence. Accessed: 16/11/2020.

Tyvold, T. S. (2018). *Short-Term Electric Load Forecasting Using Artificial Neural Networks*. NTNU.

Ørum, E., Kuivaniemi, M., Laasonen, M., Bruseth, A. I., Jansson, E. A., Danell, A., Elkington, K., and Modig, N. (2015). *Future system inertia*. ENTSO-E.

Appendix A

Derivation of backpropagation

The goal of backpropagation is to retrieve the gradient of the loss function with respect to any weight or bias in the network. To do this however, we make an assumption that the loss function may be written as an average sum of all the individual losses in a batch, ℓ. This is valid in the case of using MSE as the cost function (Nielsen, 2015). We will begin by working our way backwards in the network, starting with the partial derivatives at the output layer. The partial derivative of a loss with respect to the kth output layer activation by use of the chain rule (CR) given by

$$\frac{\partial \ell}{\partial z_k^{[L]}} \overset{\text{CR}}{=} \frac{\partial \ell}{\partial \hat{y}_k} \frac{\partial \hat{y}_k}{\partial z_k^{[L]}} = \frac{\partial \ell}{\partial \hat{y}_k} \phi'^{[L]}(z_k^{[L]}), \tag{A.1}$$

where ϕ' denotes the derivative of the output activation function. By introducing the element-wise (Hadamard) product of two vectors, equation A.1 may be written in vectorized format. I.e.,

$$\nabla_{\boldsymbol{z}^{[L]}} \ell = \nabla_{\hat{\boldsymbol{y}}} \ell \odot \phi'(\boldsymbol{z}^{[L]}). \tag{A.2}$$

This quantity is often denoted short-hand as the output error vector $\boldsymbol{\delta}^{[L]}$ (Nielsen, 2015). To derive the error vector $\boldsymbol{\delta}^{[l]}$ for any layer, we consider that for the ith hidden unit in the lth layer we have that

$$\delta_i^{[l]} = \frac{\partial \ell}{\partial a_i^{[l]}} \frac{\partial a_i^{[l]}}{\partial z_i^{[l]}} \overset{2.18}{=} \frac{\partial \ell}{\partial a_i^{[l]}} \phi'(z_i^{[l]}). \tag{A.3}$$

We can write the above as a sum of applied chain rules, by considering that the output value of the ith unit in one layer causes some change in the activation of the kth unit in the next layer. This change is given by the weight connecting those two units, thus

$$\frac{\partial \ell}{\partial a_i^{[l]}} \overset{\text{CR}}{=} \sum_{k=1}^{n^{[l+1]}} \frac{\partial \ell}{\partial z_k^{[l+1]}} \frac{\partial z_k^{[l+1]}}{\partial a_i^{[l]}} \overset{2.17}{=} \sum_{k=1}^{n^{[l+1]}} w_{ki}^{[l+1]} \delta_k^{[l+1]} = (\boldsymbol{W}_{:,i}^{[l+1]})^T \boldsymbol{\delta}^{[l+1]}, \tag{A.4}$$

where $\boldsymbol{W}_{:,i}$ denotes a vector slice of the weight matrix at the ith column. Combining equations A.3 and A.4 yields

$$\delta_i^{[l]} = (\boldsymbol{W}_{:,i}^{[l+1]})^T \boldsymbol{\delta^{l+1}} \phi'^{[l]}(z_i^{[l]}). \tag{A.5}$$

Extending the above to a vectorized version is simply including all slices of the weight matrix, which gives

$$\boldsymbol{\delta}^{[l]} = (\boldsymbol{W}^{[l+1]})^T \boldsymbol{\delta^{l+1}} \odot \phi'^{[l]}(\boldsymbol{z}^{[l]}). \tag{A.6}$$

Notice that equations A.2 and A.6 give the possibility to recursively find the gradients as far back in the network as we would like, by starting at the output layer and working our way down. These two are the first two equations needed for backpropagation.

We now have all we need to efficiently calculate all the partial derivatives of the loss with respect to any weight or bias. For a weight $w_{ki}^{[l]}$ we have

$$\frac{\partial \ell}{\partial w_{ki}^{[l]}} \overset{\text{CR}}{=} \frac{\partial \ell}{\partial z_k^{[l]}} \frac{\partial z_k^{[l]}}{\partial w_{ki}^{[l]}} \overset{\text{A.3}}{=} \delta_k^{[l]} a_i^{[l-1]}, \tag{A.7}$$

and similarly for a bias $b_j^{[l]}$ we have

$$\frac{\partial \ell}{\partial b_j^{[l]}} = \frac{\partial \ell}{\partial z_j^{[l]}} \frac{\partial z_j^{[l]}}{\partial b_j^{[l]}} = \delta_j^{[l]}. \tag{A.8}$$

Writing equation A.7 and A.8 in vectorized notation leaves us with the last two equations that the backpropagation algorithm uses. Combined with equations A.2 and A.6 the backpropagation algorithm can find all the gradients of the network by starting at the top layer

and working backwards. The four central equations are given by

$$\boldsymbol{\delta}^{[L]} = \nabla_{\hat{\boldsymbol{y}}} \ell \odot \phi'(\boldsymbol{z}^{[L]}) \tag{A.9}$$

$$\boldsymbol{\delta}^{[l]} = (\boldsymbol{W}^{[l+1]})^T \boldsymbol{\delta}^{\boldsymbol{l+1}} \odot \phi'^{[l]}(\boldsymbol{z}^{[l]}) \tag{A.10}$$

$$\nabla_{\boldsymbol{W}^{[l]}} \ell = \boldsymbol{\delta}^{[l]} (\boldsymbol{a}^{[l-1]})^T \tag{A.11}$$

$$\nabla_{\boldsymbol{b}^{[l]}} \ell = \boldsymbol{\delta}^{[l]}, \tag{A.12}$$

Appendix B

Neural network architectures

The following appendix contains a more thorough description of the hyper-parameter optimization strategy. All the architectures and hyper-parameters of the neural network models in the thesis are also given.

B.1 Hyper-parameter optimization

In this section a more detailed description of the hyper-parameter optimization is given. The **RandomSearch** class from Keras Tuner (O'Malley et al., 2019) was used for all random searches. The early-stopping (see section 2.2.1 for details) was implemented with Keras' **EarlyStopping** on the last 15% of the training data. The search parameters are shown in table B.1.

Random searches were run for all the neural network models in development (see section 4.5.2). The initial search-spaces were established based on intuition and best practice from

Table B.1: Search parameters for the random searches

Max trials	Max epochs	Executions per trial	Seed	Early-stopping
1000	150	4	13	patience: 30 min_delta: 10^{-4}

Table B.2: Initial hyper-parameter search-space of MLP-1

Hyper-parameter	Search-space	Comments
Units: $n^{[1]}, n^{[2]}$	$[72, 480]$, steps of 24	Top results generally > 120.
Activations: $\phi^{[1]}, \phi^{[2]}$	{ReLU, eLU, SeLU, tanh}	SeLU dropped.
Learning rate: η	$\{0.00001, 0.0001, 0.001, 0.01, 0.1\}$	0.00001 too small, 0.1 too big.
Dropout rates: $p^{[1]}, p^{[2]}$	$\{0.0, 0.1, 0.2, 0.3, 0.4, 0.5\}$	None.
Optimizer	{NSGD, Nadam, Adam, Adadelta, Adagrad, RMSprop}	Adadelta and Adagrad dropped.

the literature. The top 10 (by lowest validation loss) hyper-parameter configurations after a random search were then investigated. If the results hinted towards a poorly selected initial search-space (e.g., only minimum or maximums of the search-space), then the search-space was altered in the indicated direction and the random search was ran again. When the results seemed randomly distributed and did not indicate that the optimal hyper-parameters were outside of the search space, the top three combinations were tested on the development set (2019). The configuration with the most stable learning curve (validation loss versus training loss over epochs) was selected for the particular model.

The initial search space of MLP-1 is shown in table B.2. As can be seen from the search-space, the number of trials required for a complete grid search would be

$$\underbrace{18 \cdot 18}_{Uni} \cdot \underbrace{5 \cdot 5}_{Act} \cdot \underbrace{5}_{Learn} \cdot \underbrace{6 \cdot 6}_{Drop} \cdot \underbrace{6}_{Opt} = 8,748,000, \tag{B.1}$$

hence running a complete grid search for each model would be infeasible due to time constraints. Using random searches instead guarantees that each sub-space is explored even though not every single combination is tested. I.e., imagine that the number of optimal units is in the upper limit of the unit search-space; a grid search would not find this until the very end, while a random search could be able to suggest this with far fewer trials.

It was generally found that some hyper-parameter choices rarely appeared in the top results of the searches. In particular, the number of units tended to be in the top end of the search-space. Furthermore, configurations with sigmoid activation functions (see section 2.3.3) did

not seem to perform very well, hence the sigmoid function was eventually removed from all the search spaces. The same applies to SeLU, which was removed eventually as well. Since the continuous data is standardized (see section 4.3 for details), it makes intuitive sense that the sigmoid function does not work well with this type of scale (since it saturates at zero). Learning rates of 0.00001 and 0.1 seemed to be too small and too big, respectively. They were therefore removed from the search-space.

There were also clear favourites among the optimizers included in the searches. Generally, the top scores belonged to the class of optimizers with adaptive learning rates and some form of momentum (see section 2.3.5). Particularly, the best networks typically had RMSProp, Adam or Nadam. But surprisingly, the random searches of the third generation models resulted in a lot of top results with Nesterov SGD (NSGD). Models with NSGD required a slightly larger learning rate than the optimizers with adaptive learning rates. AdaDelta and AdaGrad rarely appeared in the top results and were eventually removed from the search spaces.

While the search-spaces of units, filters, kernel sizes, and the number of **Dense** and **Conv1D** layers varied for each model, activation functions, dropout rates, optimizer algorithms and learning rates stayed pretty much the same throughout the model development process. I.e., the search-spaces from table B.2 with the mentioned adjustments were also applicable to most of the later models.

The search results of the first generation models, hence the ones with the fewest number of inputs, were generally the most equally-performing. I.e., the top ten results of the random search showed no great difference in performance. As more hyper-parameters were added to the random searches by continuously increasing model complexity, performance differences were greater between the top results. It also became increasingly difficult to choose suitable search-spaces for the second and third generation models because the search results never suggested some sub-spaces being better than other sub-spaces. Note that the search parameters (table B.1) remained the same for all model generations, hence a maximum of 1000 trials were run for each model. It is therefore suspected, that the number of trials may not have been sufficiently large to get a representative result from the last generation of models.

Regardless of the difficulties of optimizing the last generation models, the hyper-parameter combinations which resulted from random searches of the still resulted in satisfactory forecast results. Hence, the hyper-parameters of the last model in development (CNN-3-2) were also used for all the expanding window test models.

B.2 Model architectures and hyper-parameters

Since the number of layers were not included in the random searches, i.e., combinations of **Conv1D**, **Dense**, **Dropout** and **MaxPooling1D**, these were determined by heuristics for each model. For instance, in the case of the CNNs with weather variables, it was hypothesized that each variable should be given its own branch to extract features from the weather input series. After feature extraction, it makes sense to combine the extracted load and weather features in a conventional manner by using a number of fully-connected (FC) layers. This is also common procedure in CNNs found in the literature (He, 2017; Tian et al., 2018). However, keep in mind that there is no way of knowing in advance if an architecture is going to perform well.

The default parameter initialization scheme in Keras for **Dense** and **Conv1D** layers, Glorot uniform (see equation 2.41), was used for all the models. The same was done for the initialization of biases, which are set to all zeros by default. There does not seem to be a agreed consensus on which initialization scheme is preferred for optimization. Glorot uniform distribution, however, is a popular choice. Using this initialization, the author did not experience any ill-behaved trials. Refer to Glorot and Bengio (2010) for details regarding Glorot uniform initialization.

From the hyper-parameter searches, the (global) learning rate was tuned through trial-and-error and by observing the learning curve for each particular model. This is the recommended strategy from Goodfellow et al. (2016, p. 291). Too high of a learning rate may cause instability observed as oscillation; while setting the learning rate to low causes the model to learn very slowly. For most models, it was found that learning rates with a order of magnitude of 0.00025 (no GPU) and 0.001 (GPU) were stable and performed well. The difference in learning rate with/without GPUs is because distributed training distributed

Table B.3: MLP-1 hyper-parameters

	Fully-connected 1	Fully-connected 2	Optimizer: Nadam
Hyper-parameters	$n^{[1]}: 168$, $p^{[1]}: 0.1$, $\phi^{[1]}: elu$	$n^{[2]}: 256$, $p^{[2]}: 0.2$, $\phi^{[2]}: tanh$	$\eta: 0.00016$, $\beta_1: 0.9$, $\beta_2: 0.999$, $\epsilon: 10^{-7}$

Table B.4: CNN-1 hyper-parameters

	Conv 1 (Load)	Conv 2 (Temperature)	FC	Optimizer (Nadam)
Hyper-parameters	$N_f^{[1]}: 32$, $N_k^{[1]}: 15$, $\phi^{[1]}: relu$, $N_p^{[1]}: 2$, $s^{[1]}: 1$, $\rho^{[1]}:$ 'valid'	$N_f^{[2]}: 16$, $N_k^{[2]}: 12$, $\phi^{[2]}: relu$, $N_p^{[2]}: 2$, $s^{[2]}: 1$, $\rho^{[2]}:$ 'valid'	$n^{[3]}: 168$, $p^{[1]}: 0.2$, $\phi^{[3]}: tanh$	$\eta: 0.00026$, $\beta_1: 0.9$, $\beta_2: 0.999$, $\epsilon: 10^{-7}$

batches equal to the batch size to each GPU for every parameter update. Hence, if the batch size is set to 32, then each GPU gets 32 samples each, so that the global batch size is 128 (with four GPus). This in turn allowed for a larger learning rate with distributed training.

Apart from the learning rate, the optimizer hyper-parameters (see section 2.3.5) were left to the Keras defaults. This was done because the Keras default optimizer hyper-parameters are usually the same as the recommended ones in the original papers. I.e., the optimizer hyper-parameters were set to $\beta = 0.99$; $\rho = 0.95$; $\epsilon = 10^{-7}$; $\beta_1 = 0.9$; and $\beta_2 = 0.999$. Changing these may offer greater stability or faster optimization, but most optimization problems were usually resolved by just adjusting the learning rate. Moreover, none of the models took long enough to run to warrant adjustment of the parameters above.

The following figures and tables provide an overview of all the neural network architectures and hyper-parameters.

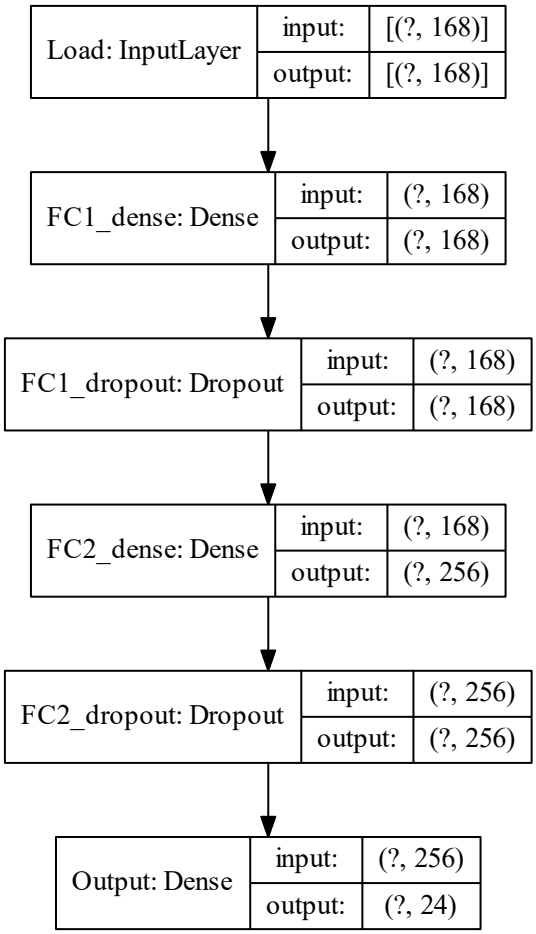


Figure B.1: MLP-1 architecture

Table B.5: CNN-2-1 hyper-parameters

	Conv 1 (Load)	Conv 2 (Temperature)	FC	Optimizer (Nadam)
Hyper-parameters	$N_f^{[11]}: 27$, $N_f^{[12]}: 45$, $N_k^{[11]}: 6$, $N_k^{[12]}: 12$, $\phi^{[11]}: tanh$, $\phi^{[12]}: tanh$, $N_p^{[11]}: 2$, $N_p^{[12]}: 1$, $s^{[11]}: 3$, $s^{[12]}: 1$, $\rho^{[11]}$: 'same' $\rho^{[12]}$: 'valid'	$N_f^{[21]}: 21$, $N_k^{[21]}: 3$, $\phi^{[21]}: tanh$, $N_p^{[21]}: 1$, $s^{[21]}: 3$, $\rho^{[21]}$: 'valid'	$n^{[3]}: 360$, $p^{[3]}: 0.3$, $\phi^{[3]}: elu$	$\eta: 0.0013$, $\beta_1: 0.9$, $\beta_2: 0.999$, $\epsilon: 10^{-7}$

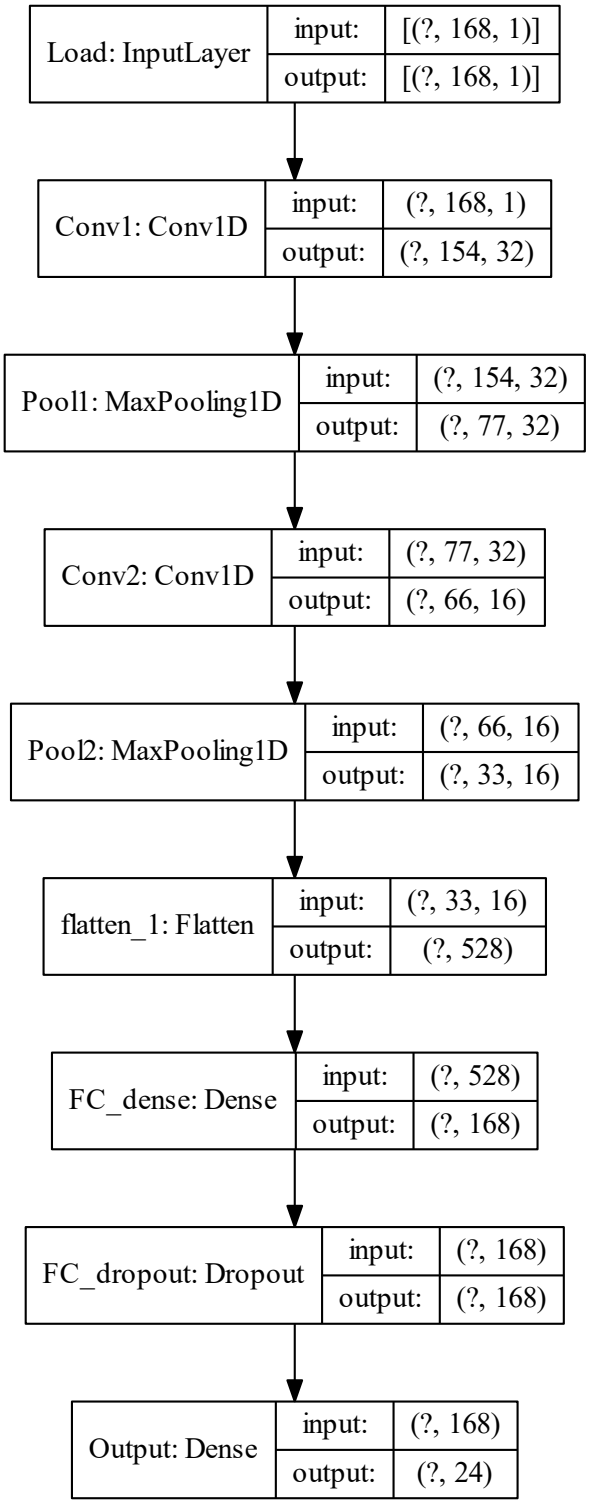


Figure B.2: CNN-1 architecture

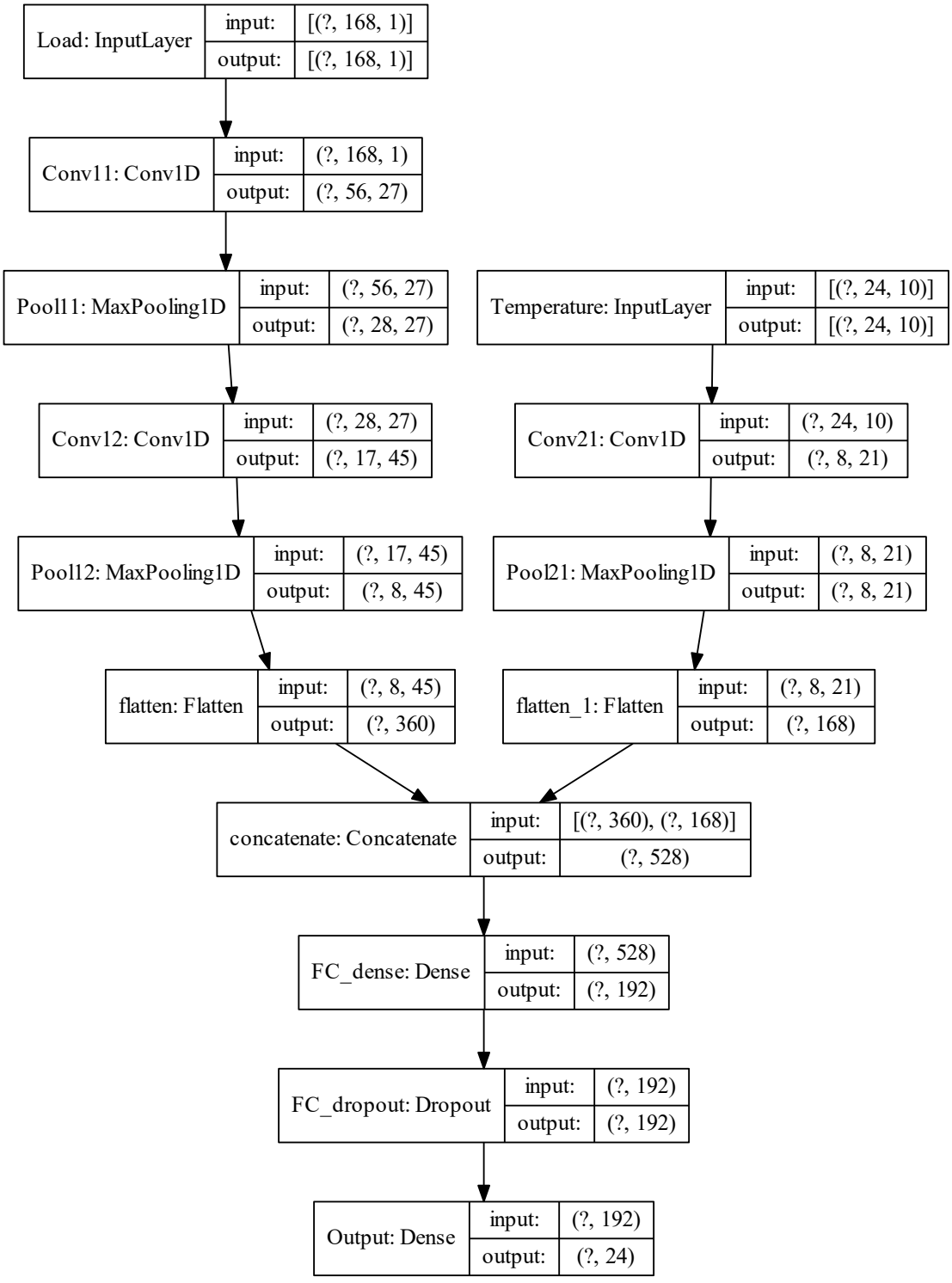


Figure B.3: CNN-2-1 architecture

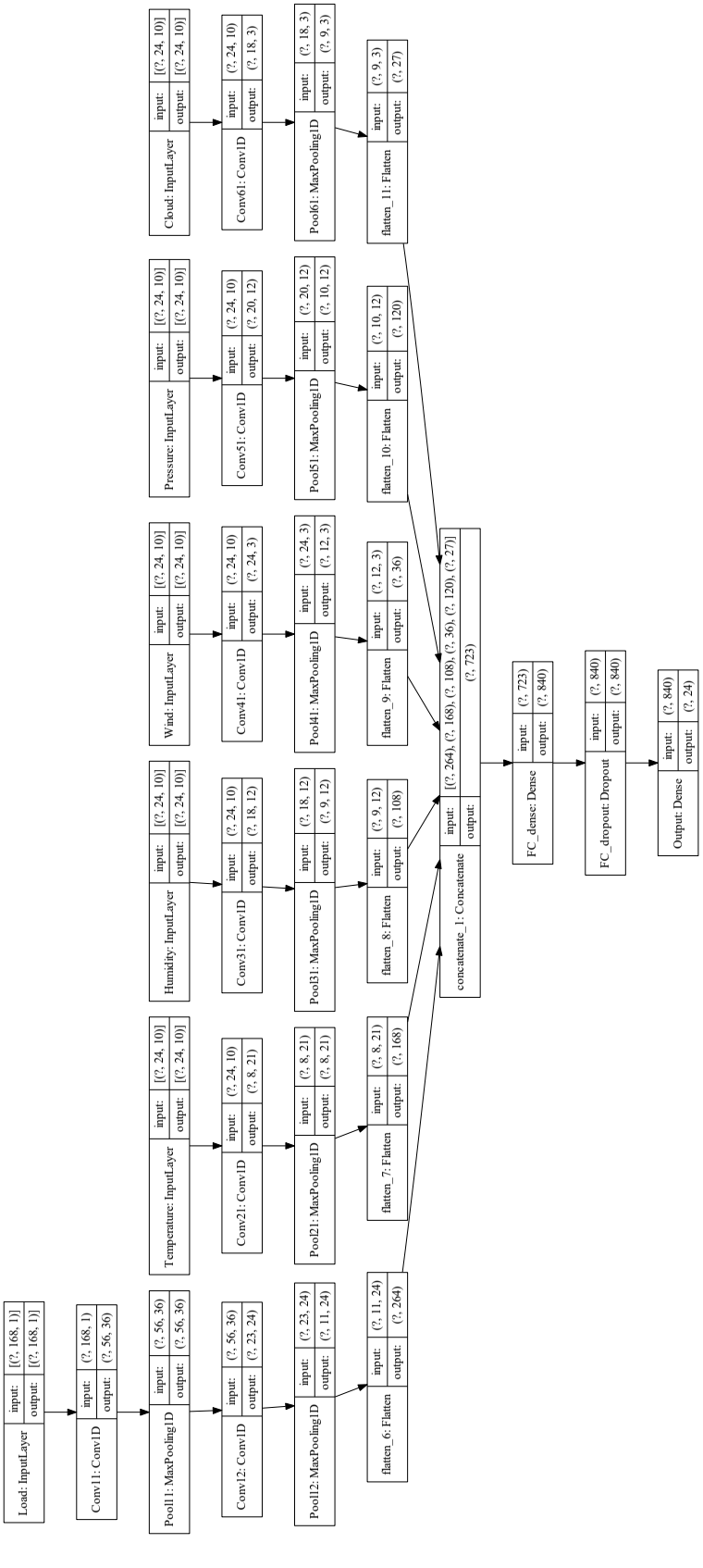


Figure B.4: CNN-2-2 architecture

Table B.6: CNN-2-2 hyper-parameters

	Conv 1 (Load)	Conv 2 (Temperature)	Conv 3 (Humidity)	Conv 4 (Pressure)	Conv 5 (Wind)	Conv 6 (Cloud)	FC	Optimizer (NSGD)
Hyper-parameters	$N_f^{[11]}$: 36, $N_f^{[12]}$: 24, $N_k^{[11]}$: 9, $N_k^{[12]}$: 12, $\phi^{[11]}$: $relu$, $\phi^{[12]}$: elu, $N_p^{[11]}$: 1, $N_p^{[12]}$: 2, $s^{[11]}$: 3, $s^{[12]}$: 2, $\rho^{[11]}$: 'same', $\rho^{[12]}$: 'valid'	$N_f^{[21]}$: 21, $N_k^{[21]}$: 3, $\phi^{[21]}$: $tanh$, $N_p^{[21]}$: 1 $s^{[21]}$: 3, $\rho^{[21]}$: 'valid'	$N_f^{[31]}$: 12, $N_k^{[31]}$: 7, $\phi^{[31]}$: $tanh$, $N_p^{[31]}$: 2 $s^{[31]}$: 1, $\rho^{[31]}$: 'valid'	$N_f^{[41]}$: 12, $N_k^{[41]}$: 5, $\phi^{[41]}$: $tanh$, $N_p^{[41]}$: 2 $s^{[41]}$: 1, $\rho^{[41]}$: 'valid'	$N_f^{[51]}$: 12, $N_k^{[51]}$: 5, $\phi^{[51]}$: $tanh$, $N_p^{[51]}$: 2 $s^{[51]}$: 1, $\rho^{[51]}$: 'valid'	$N_f^{[61]}$: 3, $N_k^{[61]}$: 7, $\phi^{[61]}$: $tanh$, $N_p^{[21]}$: 2 $s^{[61]}$: 1, $\rho^{[21]}$: 'valid'	$n^{[7]}$: 840, $p^{[7]}$: 0.3 $\phi^{[7]}$: $tanh$	η : 0.00237, β : 0.99

Table B.7: CNN-2-3 hyper-parameters

	Conv 1 (Load)	Conv 2 (Temperature)	Conv 3 (Humidity)	Conv 4 (Pressure)	FC	Optimizer (NSGD)
Hyper-parameters	$N_f^{[11]}$: 27, $N_f^{[12]}$: 45, $N_k^{[11]}$: 6, $N_k^{[12]}$: 12, $\phi^{[11]}$: $tanh$, $\phi^{[12]}$: $tanh$, $N_p^{[11]}$: 2, $N_p^{[12]}$: 2, $s^{[11]}$: 3, $s^{[12]}$: 1, $\rho^{[11]}$: 'same', $\rho^{[12]}$: 'valid'	$N_f^{[21]}$: 21, $N_k^{[21]}$: 3, $\phi^{[21]}$: $tanh$, $N_p^{[21]}$: 1 $s^{[21]}$: 3, $\rho^{[21]}$: 'valid'	$N_f^{[31]}$: 3, $N_k^{[31]}$: 1, $\phi^{[31]}$: $tanh$, $N_p^{[31]}$: 2 $s^{[31]}$: 1, $\rho^{[31]}$: 'valid'	$N_f^{[41]}$: 12, $N_k^{[41]}$: 3, $\phi^{[41]}$: $relu$, $N_p^{[41]}$: 2 $s^{[41]}$: 1, $\rho^{[41]}$: 'valid'	$n^{[5]}$: 720, $p^{[5]}$: 0.4 $\phi^{[5]}$: $tanh$	η : 0.00217, β : 0.99

Table B.8: CNN-3-1 and CNN-3-2 hyper-parameters

	Conv 1 (Load)	Conv 2 (Temperature)	Conv 3 (Humidity)	Conv 4 (Pressure)	FC	Optimizer (NSGD)
Hyper-parameters	$N_f^{[11]}$: 36, $N_f^{[12]}$: 12, $N_k^{[11]}$: 15, $N_k^{[12]}$: 12, $\phi^{[11]}$: $relu$, $\phi^{[12]}$: $tanh$, $N_p^{[11]}$: 2, $N_p^{[12]}$: 2, $s^{[11]}$: 2, $s^{[12]}$: 2, $\rho^{[11]}$: 'same', $\rho^{[12]}$: 'valid'	$N_f^{[21]}$: 21, $N_k^{[21]}$: 3, $\phi^{[21]}$: $tanh$, $N_p^{[21]}$: 1 $s^{[21]}$: 3, $\rho^{[21]}$: 'valid'	$N_f^{[31]}$: 3, $N_k^{[31]}$: 3, $\phi^{[31]}$: $tanh$, $N_p^{[31]}$: 2 $s^{[31]}$: 1, $\rho^{[31]}$: 'valid'	$N_f^{[41]}$: 12, $N_k^{[41]}$: 5, $\phi^{[41]}$: $tanh$, $N_p^{[41]}$: 1 $s^{[41]}$: 1, $\rho^{[41]}$: 'valid'	$n^{[5]}$: 840, $p^{[5]}$: 0.5 $\phi^{[5]}$: $tanh$	η : 0.0027, β : 0.99

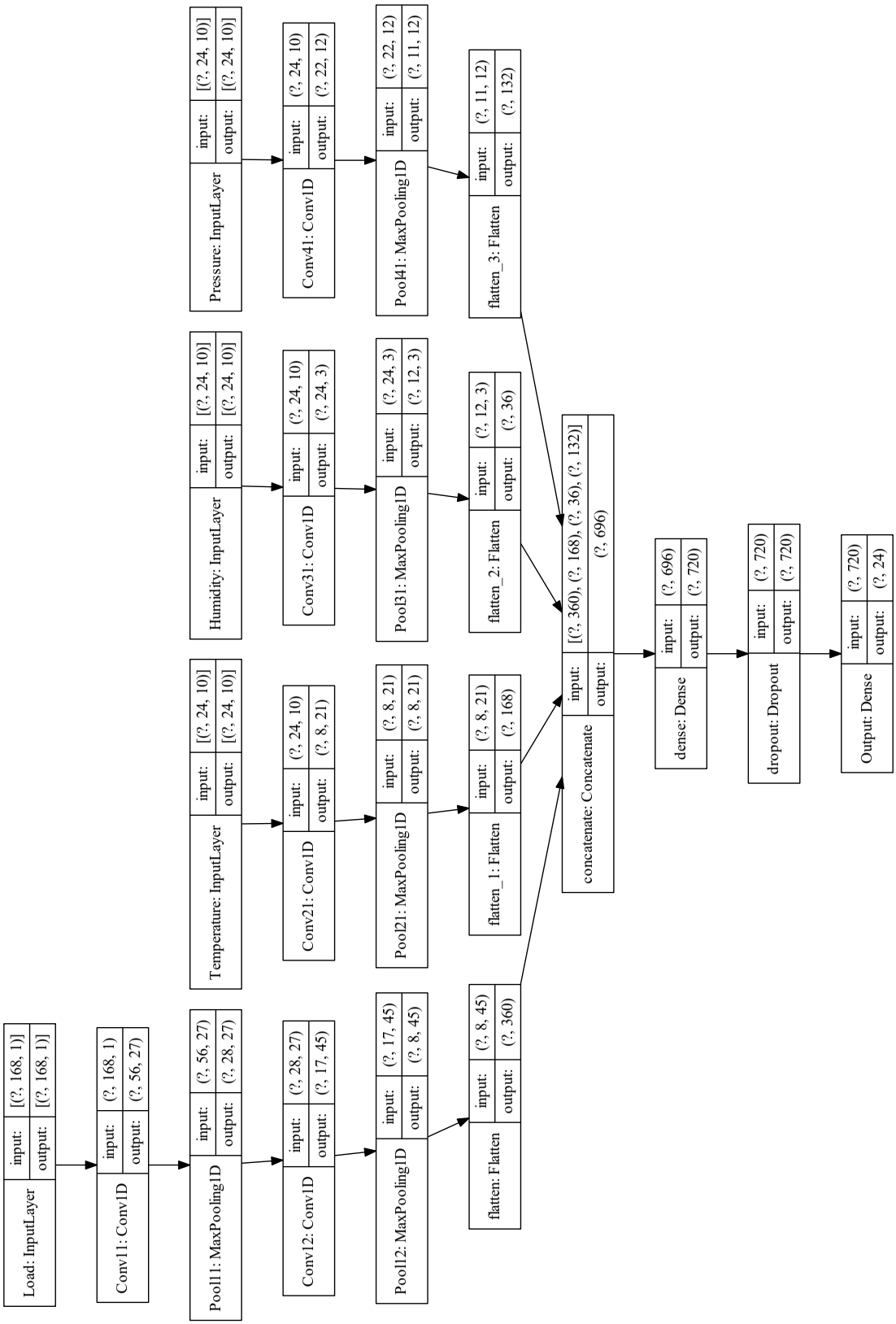


Figure B.5: CNN-2-3 architecture

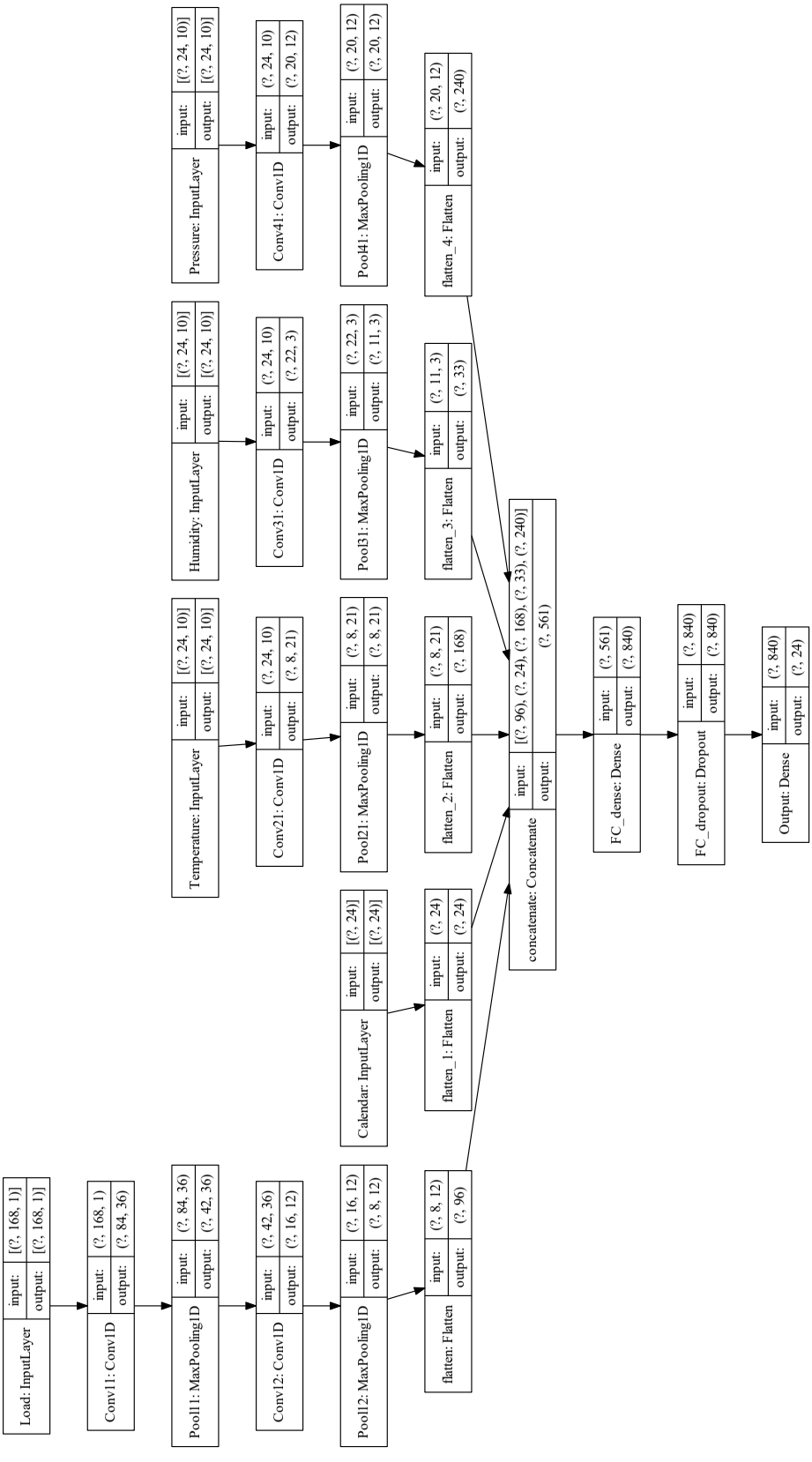
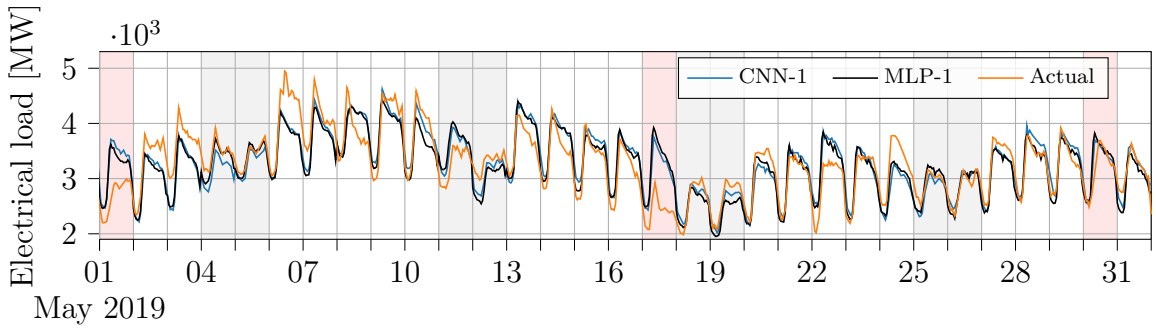


Figure B.6: CNN-3-1 and CNN-3-2 architecture

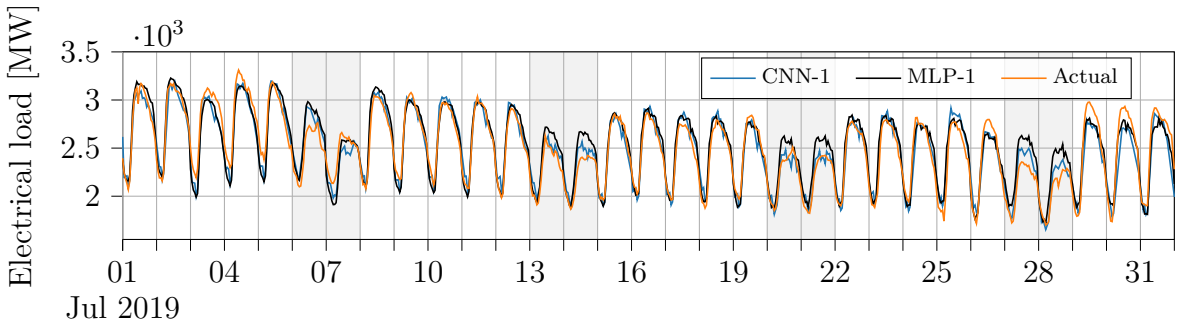
Appendix C

Model results

The following appendix contains some of the results left out from chapter 5. The best and worst seasons of the first generation and second generation models are shown in figure C.1 and C.2, respectively. The forecasts for the entirety of 2020 are given in figure C.2.

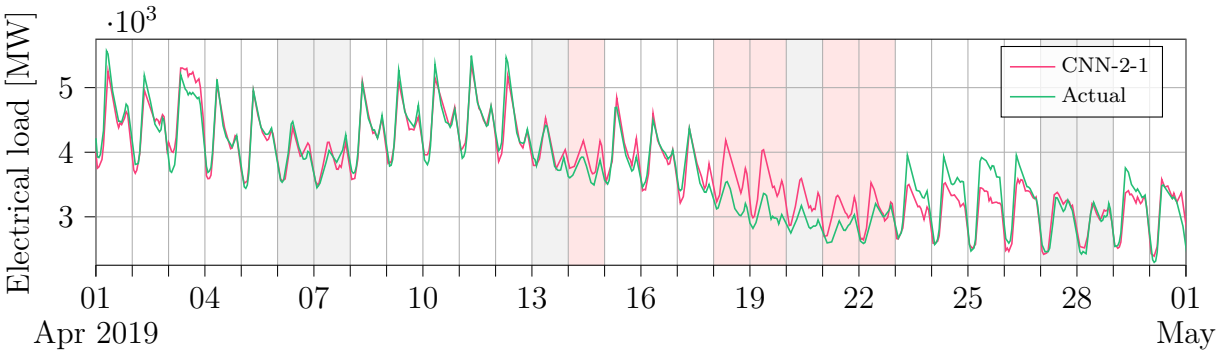


(a) Spring

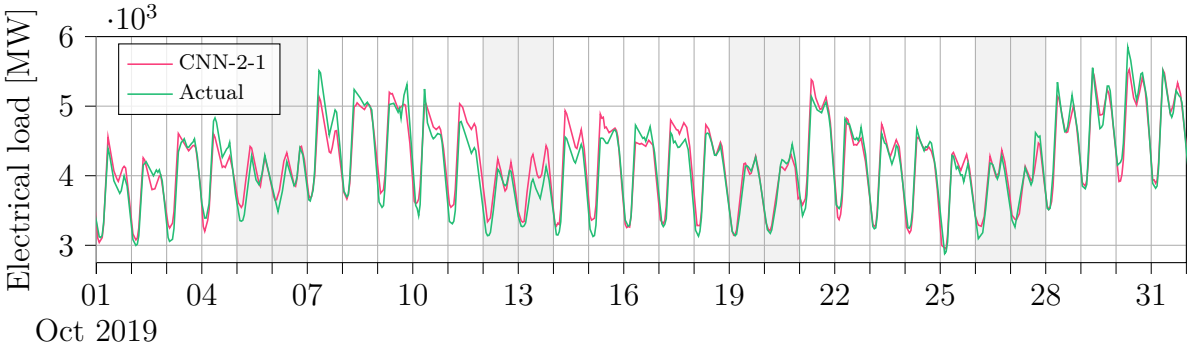


(b) Summer

Figure C.1: First generation forecasting results during the worst and best season in 2019.

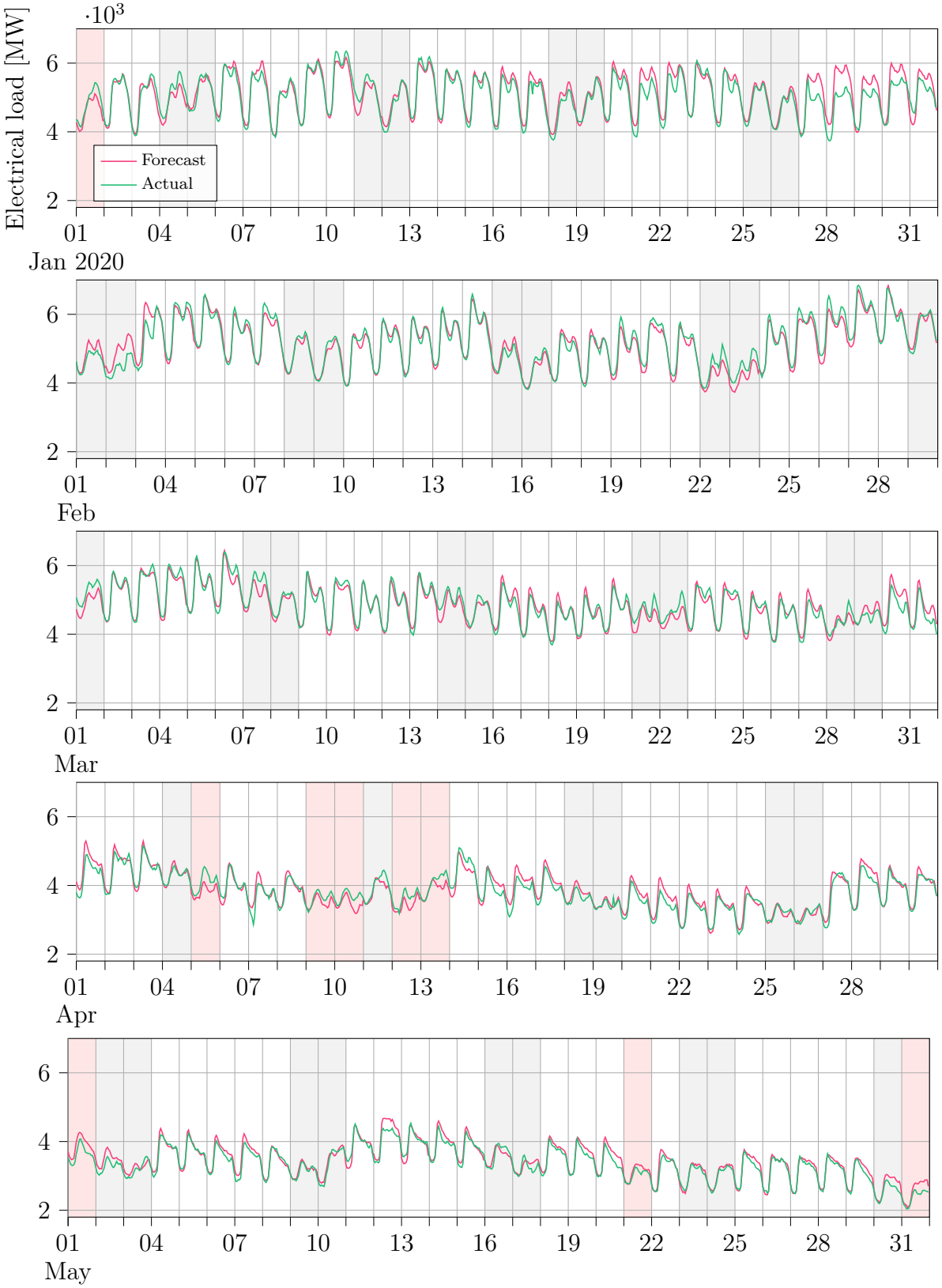


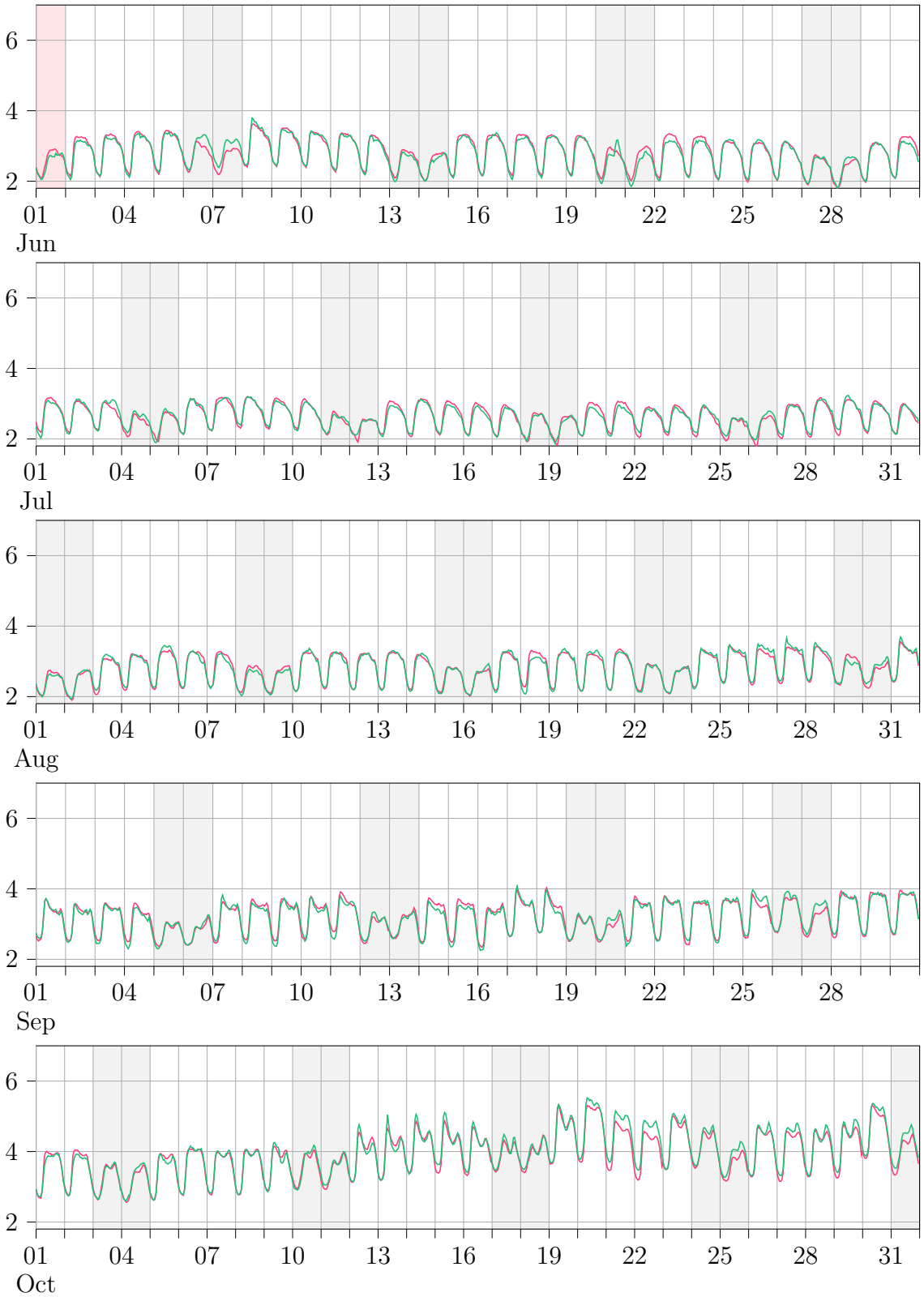
(a) Spring



(b) Autumn

Figure C.2: Second generation forecasting results during the worst and best season in 2019.


Electrical load [MW]
·10³
Forecast
Actual
Jan 2020
Feb
Mar
Apr
May


6
4
2
01
04
07
10
13
16
19
22
25
28
Jun
6
4
2
01
04
07
10
13
16
19
22
25
28
31
Jul
6
4
2
01
04
07
10
13
16
19
22
25
28
31
Aug
6
4
2
01
04
07
10
13
16
19
22
25
28
Sep
6
4
2
01
04
07
10
13
16
19
22
25
28
31
Oct

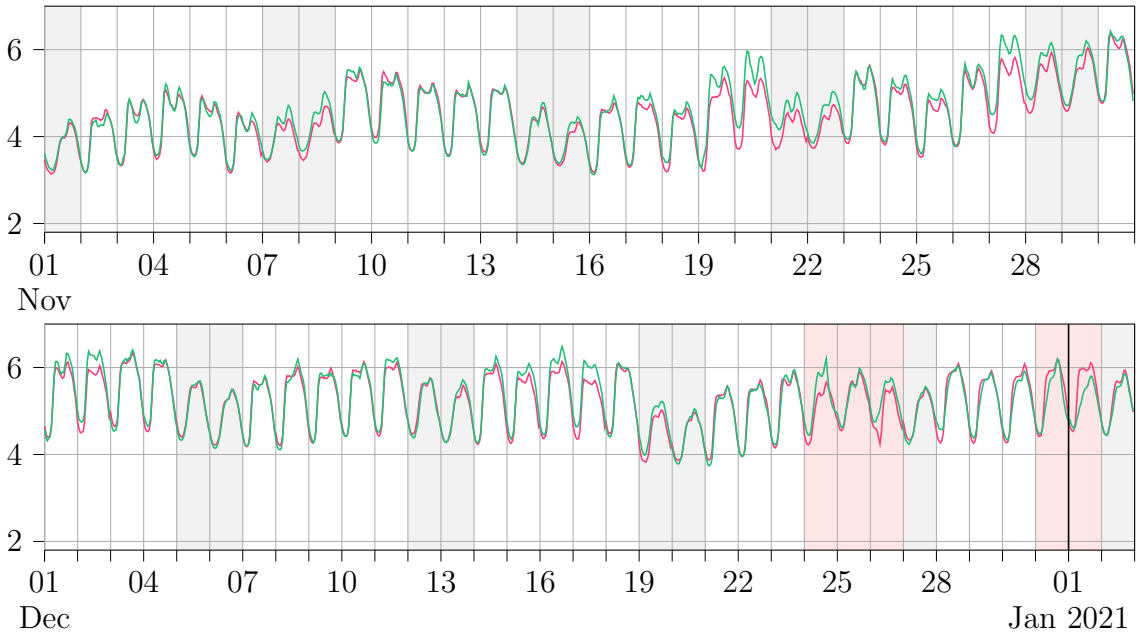


Figure C.2: The entire year of forecasts in 2020.

Daniel Aunan Bolstad

Interpretation of Electrical Load Forecasts using Explainable Artificial Intelligence

NTNU
Norwegian University of
Science and Technology