UiT The Arctic University of Norway

Faculty of Science and Technology

Department of Physics and Technology

Probabilistic Load Forecasting with Deep Conformalized Quantile Regression

–

Vilde Jensen

EOM-3901 Master’s thesis in Energy, Climate and Environment 30 SP– June 2021

Abstract

The establishment of smart grids and the introduction of distributed generation posed new challenges in energy analytics that can be tackled with machine learning algorithms. The latter, are able to handle a combination of weather and consumption data, grid measurements, and their historical records to compute inference and make predictions. An accurate energy load forecasting is essential to assure reliable grid operation and power provision at peak times when power consumption is high. However, most of the existing load forecasting algorithms provide only point estimates or probabilistic forecasting methods that construct prediction intervals without coverage guarantee. Nevertheless, information about uncertainty and prediction intervals is very useful to grid operators to evaluate the reliability of operations in the power network and to enable a risk-based strategy for configuring the grid over a conservative one.

There are two popular statistical methods used to generate prediction intervals in regression tasks: *Quantile regression* is a non-parametric probabilistic forecasting technique producing prediction intervals adaptive to local variability within the data by estimating quantile functions directly from the data. However, the actual coverage of the prediction intervals obtained via quantile regression is not guaranteed to satisfy the designed coverage level for finite samples. *Conformal prediction* is an *on-top* probabilistic forecasting framework producing symmetric prediction intervals, most often with a fixed length, guaranteed to marginally satisfy the designed coverage level for finite samples.

This thesis proposes a probabilistic load forecasting method for constructing marginally valid prediction intervals adaptive to local variability and suitable for data characterized by temporal dependencies. The method is applied in conjunction with recurrent neural networks, deep learning architectures for sequential data, which are mostly used to compute point forecasts rather than probabilistic forecasts. Specifically, the use of an ensemble of pinball-loss guided deep neural networks performing quantile regression is used together with conformal prediction to address the individual shortcomings of both techniques.

Experiments are conducted using both univariate and multivariate time series of electricity load, using two different underlying regression neural networks. The experimental results showed that the proposed method outperforms, or performs comparably to, models based on quantile regression and conformal prediction separately.

Acknowledgments

First of all, I would like to thank my supervisors Stian Normann Anfinsen and Filippo Maria Bianchi, for your guidance throughout my time working on this thesis, and for introducing me to the field of deep learning.

I would also like to thank my classmates for our collaboration and discussions, and the inspiration you have given me. Thank you, Brynhild, for our endless talks, and for always helping me find the right words.

Lastly, to my family, and to Mads: thank you for your love, support and patience.

Vilde Jensen,
Tromsø, June 2021.

Table of Contents

List of Tables

List of Figures

Abbreviations

ACF	Autocorrelation Function
Adam	Adaptive Moment Estimation
AI	Artificial Intelligence
ANN	Artificial Neural Network
AR	Autoregressive
ARIMA	Autoregressive Integrated Moving Average
ARMA	Autoregressive Moving Average
BPTT	Backpropagation Through Time
CNN	Convolutional Neural Network
CP	Conformal Prediction
GRU	Gated Recurrent Unit
KDE	Kernel Density Estimation
LSTM	Long Short-Term Memory
LTLF	Long-term Load Forecasting
MA	Moving Average
MLP	Multilayer Perceptron
MSE	Mean Squared Error
PACF	Partial Autocorrelation Function
PI	Prediction Interval
PLF	Probabilistic Load Forecasting
QR	Quantile Regression
RNN	Recurrent Neural Network
SARIMA	Seasonal Autoregressive Integrated Moving Average
SGD	Stochastic Gradient Descent
STLF	Short-term Load Forecasting
TCN	Temporal Convolutional Neural Network
TS	Time Series

Part I / Introduction

1 Motivation

Forecasting electricity load using historical observations has been of significant interest since the beginning of the electric power industry (Hong & Fan, 2016) for obvious reasons; electricity cannot be stored, only converted to other forms of energy capable of storage, and later reconverted when needed. Accurate electricity load forecasts are essential in the planning and operation of electric power systems, and can lead to substantial savings in operation and maintenance costs, and increase reliability of the power supply system (Almeshaiei & Soltan, 2011). A perfect equilibrium between electricity production and consumption is not achievable in real-world situations, but a reasonable balance between the two must be obtained to ensure that the operational limits of the electricity grid are not exceeded (Infield & Freris, 2009), as well minimizing the cost of under- and overproduction. In the electricity market, electricity is sold via bidding, where the sellers and buyers have to produce and buy the agreed-upon amount (Dalal, Mølnå, Herrem, Røen, & Gundersen, 2020), and electricity overproduction and underproduction can therefore cause financial loss to both electricity buyers and sellers. Based on the information above, it is clear that not only are the forecasts themselves vital for electricity production management, but it is also necessary to quantify the uncertainty in the forecasts. Forecasts that are able to express these uncertainties are termed *probabilistic* forecasts.

Unlike point forecasts, probabilistic load forecasts provide predictions in the form of intervals, quantiles or density functions (Hong et al., 2016), indicating the possible volatility of future demand. Prediction intervals are constructed using available past observations, often together with explanatory variables, to provide a possible range of values for a future observation, based on a given confidence level. A prediction interval is termed valid if the actual coverage of future observations equals the designed confidence level (Xu & Xie, 2020), and valid prediction intervals are essential in high-risk situations, enabling risk-based specification and operation of the network. The width of the prediction interval serves as a reliability measure of the forecast, where wider intervals indicate higher uncertainties (Quan, Srinivasan, & Khosravi, 2013). If probabilistic forecasts are unreliable or overly wide, they become ineffective, and the construction of narrow, valid, or near valid, prediction intervals is therefore paramount.

Conformal prediction (Shafer & Vovk, 2008) is a probabilistic forecasting technique that constructs distribution-free prediction intervals that attain valid marginal[1] coverage in finite samples (Romano, Patterson, & Candès, 2019). The earliest literature presents conformal prediction as a transductive method, resulting in the method being computationally expensive and impractical to use in most settings (Papadopoulos, 2008). However,

[1]Coverage probability can be separated between marginal and conditional probability. The former being the probability distribution over a subset of variables, whereas for conditional probability, one are interested in the probability of some event, given that some other event has happened (Goodfellow, Bengio, & Courville, 2016). Marginal probability considers the union of all events of the variables in the subset, rather than the probability of a single event.

the introduction of split conformal prediction (Papadopoulos, Proedrou, Vovk, & Gammerman, 2002), a method based on inductive inference, has significantly lightened the computational burden of the original method. The inductive conformal prediction framework can be wrapped around most machine learning algorithms, resulting in a broad area of use. Despite this appeal, conformal predictors assume exchangeability between samples, a statistical property closely related to the assumption of independent and identically distributed samples (Bernardo, 1996), requires data-splitting, and the resulting intervals can be unnecessarily conservative due to their constant or weakly varying length (Romano, Patterson, & Candès, 2019). The exchangeability assumption, where the information provided by the samples is assumed to be independent of the order in which the samples are collected, makes conformal prediction unsuitable for time series data, where the order of the time steps cannot be exchanged.

Another probabilistic forecasting technique is the classical *quantile regression* (Koenker & Bassett Jr, 1978), where prediction intervals are constructed by estimating two conditional quantile functions from the available data. The width of the intervals constructed using quantile regression depends on the individual observations (Romano, Patterson, & Candès, 2019) and can, therefore, significantly vary from sample to sample, allowing the model to adapt to local variability within the data, i.e. heteroscedastic data (Dutilleul & Legendre, 1993). However, the prediction intervals constructed using quantile regression are not guaranteed to satisfy the designed coverage level for finite samples, because they are only estimates of the true intervals, obtained via quantile functions estimated from the available data.

Electricity load data displays strong temporal variations and is known to exhibit non-linear dependencies (Yang, Wu, Chen, & Li, 2013), making the task of forecasting electricity load complex. A forecasting model must not only identify and learn the temporal dependencies within the data, but also grasp how and to what extent the load is affected by external factors, if included in the model. Traditional time series models such as the autoregressive (AR) model, the autoregressive integrated moving average (ARIMA) model, and their multivariate extensions, have long been used for electricity load forecasting (Dang-Ha, Bianchi, & Olsson, 2017). Statistical methods have the advantage of easily constructing prediction intervals, since they commonly assume the samples and errors of time series to be normally distributed. However, these methods are unable to model non-linear dependencies in the data and make strong assumptions of the distribution of the underlying data generation process, which often are unknown (Box, Jenkins, Reinsel, & Ljung, 2015).

Due to the limitations of the classical parametric time series models such as AR and ARIMA, machine learning methods, and especially deep neural networks, have been proposed to solve the electricity load forecasting problem. Deep neural networks have the advantage of automatic learning non-linear relationships without the need for significant prior knowledge about the distribution of the data, and require less data preprocessing compared to the statistical models (Gasthaus et al., 2019). However, neural networks generally produces forecasts in the form of point estimates (Keren, Cummins, & Schuller, 2018). The aforementioned practical and beneficial properties of neural networks motivates the core of this thesis; to equip deep learning methods with the ability of providing forecasts in the form of valid prediction intervals, suited for the task of electricity load forecasting.

2 Introduction to Machine Learning

Machine learning (ML), a core component in the field of artificial intelligence (AI), is a discipline focused on the construction of algorithms and models able to automatically learn and improve through experience (Jordan & Mitchell, 2015), extracting knowledge directly from raw data, without being specifically instructed on how to do so. The term *learning* in machine learning refers to the problem of improving some performance measure when performing a task, where the improvement is based on gradually optimizing an objective function on a provided training dataset (Goodfellow et al., 2016). On a high level, machine learning systems consist of a data source from which training data is extracted, which then is inserted into a model to produce an output. A measure of error is found using a function that quantifies the performance on a downstream task into a numerical value, or *cost*. The model is corrected according to the error measurement, where the goal commonly is to minimize the cost value, thereby improving the model's performance. The process of calculating the error and correcting the model is repeated until a specified performance level is obtained, or the model performance on a validation dataset stops increasing.

Machine learning methods can be utilized to solve both *supervised* and *unsupervised* learning problems, the former being the most common problem setting (Jordan & Mitchell, 2015). In supervised learning, the learning algorithms are presented with data with samples in the form of inputs and corresponding output, often referred to as *observations* and *labels*, respectively (Theodoridis & Koutroumbas, 2009). The supervised algorithms are trained to create a mapping of the input-output relationship, that is, to reproduce the correct output given a specific input, and learns from measuring the error between the estimated labels and the true labels, updating accordingly. In unsupervised learning, the algorithms are presented with inputs alone, i.e. *unlabeled* data. The absence of labels changes the objective of the learning problem from learning the input-output mapping, to learning natural groupings, structures, or patterns within the training data, as done in for example clustering problems (Caron, Bojanowski, Joulin, & Douze, 2018).

Deep learning, a field within machine learning, uses deep neural networks to solve learning problems. The core idea of these networks is the use of many simple constituents that together form a sophisticated model (Anthony & Bartlett, 2009); Deep neural networks consist of stacked layers containing computing elements that use a mathematical function to map input values into output values. The output of computing elements in one layer is used as input to the elements in the subsequent layers, and possibly also to preceding layers, if the network architecture is recurrent. The use of several such layers increases the depth of the network, making them *deep*, resulting in more mappings taking place and enabling the networks to perform more complex tasks. At each layer, the mapping extracts features of the input, and having deeper networks allows finer and more detailed features to be extracted at the later layers, increasing the network's capability.

The main drivers behind the development of machine learning are the rapid growth in the ability to collect, process, and store vast amounts of data, and as a result, the increased availability of such data (Jordan & Mitchell, 2015). Digitalization has resulted in modern datasets being too large for manual analysis, making the development of algorithms capable of efficiently handling large-scale data essential. The earliest developments of machine learning algorithms showed results far less accurate and valuable than what we see today. The fundamental reason for this is that we are now able to provide these algorithms with the resources needed to improve the performance; enough data and computational power. Machine learning algorithms aim to create a model based on a provided training dataset, that generalizes well, thereby producing satisfactory results on new, unseen data (Goodfellow et al., 2016). Using a collection of samples to make inference about a whole population generally only produces sensible results if the collection of samples is large and representative enough (Casella & Berger, 2021), and substantial amounts of data are therefore needed.

3 Introduction to Electricity Load Forecasting

Electricity load forecasting refers to forecasting the expected electricity demand and is vital for planning and operation in the power industry. The power sector has in recent time undergone a rapid change due to the deployment of smart grid technology and integration of renewable energy sources (Hong & Fan, 2016). Due to the intermittent nature of most renewable energy sources, e.g. wind and solar power, their integration has given rise to several challenges for the operational reliability of the electric grid on both the production and consumption side (Taylor & McSharry, 2007). The challenges lie in the difficulty of matching the irregular energy production from the renewable sources to the constantly varying electricity need of the consumers. Therefore, forecasting the expected electricity load is crucial.

The task of electricity load forecasting is not straightforward; there are several factors that impact electricity consumption, e.g. climatic conditions, where temperature poses the most significant influence, customer activities, and holidays, resulting in a complex and dynamic system. Electricity load data display diverse and complex patterns, as well as unpredictable components driven by social and environmental factors (Almeshaiei & Soltan, 2011). The degree to which the electricity load is affected by different external factors is highly varying from case to case, e.g. the influence of temperature on yearly electricity usage for a household located in Norway will significantly differ from a household located in Southern Europe, due to the seasonal temperature difference between the two countries. Additionally, electricity load has a strong cyclic time dependence (Yang et al., 2013), i.e. the load for a given hour is dependent not only on the load of the most recent preceding hours, but also on the same hours in preceding weeks, which further have additional dependencies.

Electricity load forecasting can be categorized based on the horizon of the predictions, where the two main categories are short-term load forecasting (STLF) and long-term load forecasting (LTLF), with a cut-off horizon of two weeks (Hong & Fan, 2016). Load forecasting is not only differentiated based on the forecasting horizon but also on the forecasting level, where the levels range from household to industrial level. Oftentimes, several individual electricity consumers are aggregated together into one group, and forecasts are constructed at aggregated levels. Forecasting the energy consumption of individual households is strenuous compared to forecasting at aggregated levels due to the increased fluctuations exhibited in signal patterns at the lower levels (Gasparin, Lukovic, & Alippi, 2019), whereas when load consumption is aggregated, the variation within the signal decreases, and the signal patterns change in a slower manner. Based on the information presented above, it becomes clear that no single model can be generalized to perform well for all cases, and individual models must be constructed based on characteristics such as temporal resolution, customer consumption level, and customer location, to name a few.

The majority of the existing load forecasting methods, both statistical and based on deep learning, produces point estimates, i.e., a single value for each step in the forecasting horizon, where the outputs commonly are the expected value or the conditional mean of future load (Chen, Kang, Chen, & Wang, 2020; Hong & Fan, 2016). On the other hand, probabilistic methods produce predictions with an associated confidence. The probabilistic forecasts are able to better model the uncertainty of the future, compared to the traditional point forecasts, which is essential in the planning and operation of the power sector industry. Knowing the uncertainty of the forecasts is of particular significance since erroneous electricity load forecasts can have major cost implications for companies operating in competitive power markets (Taylor & McSharry, 2007).

The most commonly used techniques in the field of electricity load forecasting can be roughly grouped into two categories; statistical-based methods and artificial intelligence-based methods (Hong & Fan, 2016). Statistical autoregressive models have for many years been dominating the field of time series forecasting (Dang-Ha et al., 2017; Brownlee, 2018), but in recent years the use of neural networks has shown promising results. Specifically, recurrent neural networks (RNNs) and convolutional neural networks (CNNs) have been of main interest, due to their effective ability to process and extract information from a large history of input data in a nonlinear manner. RNNs are one of the most frequently used deep neural network for time series forecasting (Chen et al., 2020), due to their internal recurrent connections which enable the networks to acquire a memory of previous input values, making them suitable for data containing temporal dependencies. Recently, a specialized type of convolutional neural network named temporal convolutional network (Lea, Flynn, Vidal, Reiter, & Hager, 2017) has gained increasing popularity within the field of time series forecasting, due to their long effective memory, and small number of trainable network parameters.

4 Research Questions, Proposed Approach, and Contributions

Based on the motivational factors presented in Section 1, the following research questions can be formulated:

- *Is it possible to combine conformal prediction and quantile regression to construct a probabilistic forecasting method that inherits the advantages of both techniques?*

- *Can conformal predictors be applied to the problem of time series forecasting while preserving the temporal ordering of the observations?*

This thesis proposes a probabilistic electricity load forecasting method based on conformal prediction and quantile regression. Specifically, the proposed method uses an ensemble of deep neural networks optimized by the pinball loss function to extend point forecasts into probabilistic forecasts in the form of quantiles, forming the upper and lower bound of a prediction interval. The prediction interval obtained using quantile regression is adjusted by using conformal prediction to better the prediction interval's coverage and width. Experiments are conducted using univariate and multivariate time series data containing observations of electricity load both at individual household level and at aggregated levels for household, industry, and cabin users. The experimental results illustrate the promising potential and strength of the proposed approach.

The key contributions of this thesis can be summarized as follows:

- Proposing a distribution-free probabilistic forecasting method for constructing approximately marginal valid prediction intervals adaptive to heteroscedastic time series.

- Verifying the potential of the proposed method using univariate and multivariate electricity load datasets, where the proposed method achieves equal or superior performance compared to commonly used statistical and artificial intelligence-based probabilistic forecasting methods.

5 Thesis Outline

This thesis consists of five main parts. After the introduction in Part I, relevant technical background information of concepts related to the content of the thesis is summarized in Part II. Section 6 introduces the problem of time series forecasting, probabilistic forecasting, and time series forecasting models. In Section 7, concepts within machine learning and artificial neural networks are presented. Section 8 and 9 presents theory of two distribution-free probabilistic forecasting methodologies, namely quantile regression and conformal prediction, respectively.

Part III presents the proposed method and closely related works.

The experiments conducted to assess the performance of the proposed method are covered in Part IV, describing the datasets and models used, and presenting and discussing the experimental results.

Lastly, concluding remarks and ideas and directions for future work are presented in Part V.

Part II / Technical Background

6 Time Series Forecasting

A time series can be defined as a collection of random variables indexed according to the order they are obtained in time (Shumway & Stoffer, 2017). By including the time at which a sample is obtained, additional knowledge and insight about the data-generating process can be gained. The process of extracting knowledge about possibly useful patterns, seasonalities, and trends in the data, is called time series analysis. Time series forecasting aims at constructing a mathematical model that can be used to predict future values of a time series. The forecasting models are typically based on historical records of the time series and, in some cases, exogenous variables (covariates) are included. The exogenous variables are often presented as additional, separate time series.

The motivating factor for including multiple variables in time series forecasting is that the aggregated information obtained from all time series can be useful when making predictions, since time series may have similar characteristics, and the variation within one time series may be dependent on the variation within another. By including exogenous variables that are known to explain the values of the time series being predicted, the accuracy of the predictions can be improved. A time series that includes records of only one variable is called univariate, whereas a time series that includes several variables is called multivariate. Oftentimes, there exist many strongly correlated time series. Including them into a forecasting model allows the model to learn long-term temporal dependencies between series, which can improve the robustness of the model as well as the quality of the forecasts (Borovykh, Bohte, & Oosterlee, 2017). In multivariate time series forecasting, historical values of the exogenous variables can be included, as well as the future values of these variables, if available, either as observations or via predictions. For example, when predicting electricity load, which is highly dependent on temperature, both past and future weather forecasts can be included in the forecasting model.

(Yule, 1927) postulated that a time series can be regarded as a realization of a stochastic process, making the objective of time series forecasting to model the underlying process that generates the observed values of the time series, i.e. finding the function that best predicts the true future values of the time series. The optimal forecasting model is the one that minimizes the error between the true and the predicted values of the time series. For a multivariate time series, a multi-step forecasting problem can be expressed in the following way:

$$y_{(t+1):(t+T)} = f\left(y_{1:t}, X^{(i)}_{1:t}, X^{(i)}_{(t+1):(t+T)}\right), \quad i = 1, .., N \tag{1}$$

where y and $X^{(i)}$ denote the variable whose future values are to be predicted and the exogenous variables, respectively. *1:t* is the number of time steps for which historical observations are available, and T is the length of the forecasting horizon. The function f is the forecasting model, explaining the dependency relationship between the past and future values of the time series, as well as the relationship between the dependent variable and the covariates.

6.1 Single-step Forecasting

Single-step or one-step-ahead forecasts are predictions made about the next subsequent time step $(t+1)$. For single-step forecast, $T = 1$ and Eq. (1) reduces to

$$y_{(t+1)} = f(y_{1:t}, X_{1:t}^{(i)}, X_{(t+1)}^{(i)}), \quad i = 1, .., N.$$

A single-step forecast can be the prediction of the next minute, hour, or day, depending on the temporal resolution of the time series whose values are being predicted, and for electricity load forecasting, short-term load forecasting often involves single-step forecasts.

6.2 Multi-step Forecasting

When the length of the forecasting horizon exceeds a single step, such that $T > 1$, the predictions are called multi-step forecasts. Multi-step forecasting tasks are more challenging compared to single-step forecasting due to increased uncertainty and error accumulation, often leading to reduced accuracy (Masum, Liu, & Chiverton, 2018). There are multiple ways of constructing multi-step forecasts, the two most common being the *direct* and *recursive* methods. A recursive strategy creates one-step-ahead predictions for each time step in the forecasting horizon, feeding the forecasts back to the model as an input to create the next one-step-ahead forecast. This iterative strategy uses the same forecasting model T times to create the multi-step forecast at $(t+T)$. The most obvious problem with this strategy is that since the forecasts are fed back to the model as input, the prediction error accumulates at each step of the recursion.

The direct forecasting strategy creates an individual forecasting model for each step in the forecasting horizon, utilizing a total of T models to construct the T forecasts at $t+1$ to $t+T$. The direct strategy avoids the error accumulation of the recursive method, but since the forecasts are constructed using separate models, this method does not guarantee statistical dependence between the forecasts. Combinations of the two methods have been suggested, and for a more in-depth review of different multi-step forecasting strategies, the reader is referred to (Taieb, Bontempi, Atiya, & Sorjamaa, 2012).

6.3 Time Series Forecasting as a Supervised Learning Problem

A time series consists of successive observations, and univariate and multivariate time series are represented as vectors or matrices of observations, respectively. Time series forecasting can be cast as a supervised learning problem by transforming the observations into input-output pairs, and the forecasting model is trained to learn to map the inputs to the outputs. The input and output components each consists of a chosen number of observations, where the observations in the input component are past time steps used to predict the future time steps contained in the output component. This can for a univariate time series be illustrated in the simple example below:

$$\begin{array}{ll}
\text{Time series : } [1,2,3,4,5,6] & \\
\text{Input} & \text{Output} \\
[1,2,3] & [4] \\
[2,3,4] & [5] \\
[3,4,5] & [6]
\end{array}$$

In this example, a univariate time series with six observations is transformed into a two-dimensional structure with three samples, where three previous time steps are used to make a *one-step-ahead* prediction. This type of transformation of the observations is called the *sliding* or *rolling window method* (Brownlee, 2018). The training data is created by sliding a window with fixed size over the time series, moving it one time step at a time. The width of the window can be changed to include more or fewer observations. If the output component contains more than one observation, one refers to the problem as a multi-step forecasting problem, where two or more future time steps are to be predicted.

6.4 Probabilistic Forecasting

Forecasts can be constructed as a single value for each step in the forecasting horizon, e.g. the expected value, and are then referred to as point forecasts. Point forecasts do not indicate the certainty of prediction and, in some situations, having only the point forecasts is enough. In other situations, such as planning and decision-making processes, the certainty of the forecasts is paramount and probabilistic forecasts are preferred. Probabilistic forecasting models complement the point forecasts with quantiles, intervals, or density functions, enabling the forecaster to assess the uncertainty of the predictions made.

In the field of electricity load forecasting, the main focus within the literature has been on point forecasting (Liu, Nowotarski, Hong, & Weron, 2017; Wang, Zhang, et al., 2019), while less attention has been devoted to probabilistic load forecasts. (Hong & Fan, 2016) presents a tutorial review on probabilistic load forecasting, which covers various techniques for both short-term and long-term load forecasting. They report that there has been a significant increase in the amount of published literature on probabilistic load forecasting in recent years, but that there are many errors and inconsistencies within the area, e.g. constructing probabilistic forecasts without true probabilistic meaning, and wrongly evaluating probabilistic forecasts. There are several ways to construct probabilistic forecasts, and the authors categorize probabilistic load forecasting techniques into two gropes based on their original purpose (Hong & Fan, 2016):

1. **Constructing probabilistic forecasts via the extension of point forecasts.** Techniques originally designed for point forecasts can be extended to construct prediction intervals used to quantify the uncertainty of the original point forecast. The extension can be performed via post-processing of the point forecasts, or using *on-top* frameworks. This is further discussed in Section 6.5.2.1.

2. **Constructing probabilistic forecasts using techniques developed with the probabilistic forecasting feature.** Methods such as non-parametric probability density estimation, Bayesian models, and quantile regression constructs forecasts with an associated confidence level directly, and are therefore intrinsically probabilistic. Such techniques are throughout the rest of this thesis referred to as direct probabilistic forecasting techniques.

6.4.1 Prediction Intervals

The most common way of presenting probabilistic forecasts is the extension from point forecasts using prediction intervals (Nowotarski & Weron, 2018). These prediction intervals provide an upper and lower limit where a future value is expected to lie with a given probability, or confidence level, α. Prediction intervals can be defined as follows:

Let $X \in \mathcal{X}$ and $Y \in \mathcal{Y}$ be random variables representing the input observation and label, respectively, following a joint distribution $\pi(X,Y)$, where $\pi(Y|X)$ denotes the conditional distribution of Y given X. A prediction interval constructed using a collection of training samples $\{(x_i, y_i), i = 1, 2, \dots, n\}$, where (x_i, y_i) are realizations of (X, Y), is given by $\hat{C}_{\pi,n}(X) = [L(x), U(x)]$, where L and U are functions that map from $\mathcal{X}$ to $\mathcal{Y}$. A prediction interval is called valid, or calibrated, if the interval coverage of a new test point $(X_{n+1}, Y_{n+1}) \sim \pi$ is guaranteed to be greater or equal to the designed confidence level, where the coverage of a prediction interval refers to if the interval contains the actual value of the predicted variable, or not. Prediction interval coverage guarantees are further discussed in the following subsection. The width[2], or length, of a prediction interval, $U(X_{n+1}) - L(X_{n+1})$, is governed by the confidence level, i.e. the probability of a new observation lying within the provided prediction interval, where more uncertain predictions produce wide intervals. When prediction intervals become very wide, they tend to be less informative and denote high uncertainty in the prediction model.

(Gneiting & Katzfuss, 2014) argue that the ideal prediction interval must maximize the sharpness of the predictive distributions, subject to calibration. The sharpness of probabilistic forecasts refers to the concentration of the predictive distributions, i.e. how tightly the prediction interval covers the actual distribution. Thus, probabilistic forecasts in the form of intervals should be as narrow as possible while reflecting the designed confidence level. Additionally, an ideal procedure for constructing prediction intervals should make no strong assumptions on the underlying data distribution (Romano, Patterson, & Candès, 2019), since it is often unknown.

6.4.1.1 Marginal and Conditional Coverage

Prediction interval coverage guarantees can be defined on average over a set of test points, or pointwise for any fixed value $X_{n+1} = x$, termed marginal and conditional coverage

[2]Prediction interval width is often also referred to as prediction interval length, or prediction interval sharpness. Throughout the thesis, these terms are used interchangeably, and refers to the difference between the upper and lower interval bounds.

guarantee, respectively (Barber, Candes, Ramdas, & Tibshirani, 2019). For a distribution-free marginal coverage guarantee, the probability that the prediction interval covers the true test value Y_{n+1} must be at least $1 - \alpha$ on average over a random draw of training and test data from any underlying distribution π:

$$\mathbb{P}\{Y_{n+1} \in \hat{C}_{\pi,n}(X_{n+1})\} \geq 1 - \alpha\,. \tag{2}$$

Conditional coverage is a much stricter definition compared to marginal coverage, and hence harder to obtain. A prediction interval satisfies conditional coverage on the $1 - \alpha$ level if

$$\mathbb{P}\{Y_{n+1} \in \hat{C}_{\pi,n}(X_{n+1}) | X_{n+1} = x\} \geq 1 - \alpha\,, \tag{3}$$

meaning that for any point x, the probability that $\hat{C}_{\pi,n}$ covers $X_{n+1} = x$ must at least be $1 - \alpha$. To demonstrate the difference between marginal and conditional coverage, (Barber et al., 2019) presents the following example:

> *As a motivating example, suppose that each data point i corresponds to a patient, with X_i encoding relevant covariates (age, family history, current symptoms, etc.), while the response Y_i measures a quantitative outcome (e.g., reduction in blood pressure after treatment with a drug). When a new patient arrives at the doctor's office with covariate values X_{n+1}, the doctor would like to be able to predict their eventual outcome Y_{n+1} with a range, making a statement along the lines of: "Based on your age, family history, and current symptoms, you can expect your blood pressure to go down by 10–15 mmHg".*

When setting $\alpha = 0.05$, the statement made by the doctor should hold with a probability of 95%. For marginal coverage, the statement has a 95% probability of being accurate on average for all possible patients. However, the statement might have a significantly lower, or even 0%, chance of being accurate for patients of specific age groups, being averaged out by a higher-than-95% coverage probability of the other age groups. For conditional coverage, the statement made by the doctor must hold with 95% probability for every individual patient, regardless of age. Therefore, conditional coverage is more difficult to ensure.

Due to the stricter requirements, conditional coverage could be impossible to satisfy in distribution-free settings (Xu & Xie, 2020). Consequently, most probabilistic forecasting methods focus on satisfying marginal coverage, or a compromise between marginal and conditional coverage. Having said that, marginal coverage-focused methods can possibly, but not necessarily, obtain conditional coverage as well.

6.5 Time Series Forecasting Models

This subsection introduces commonly utilized statistical and neural network-based time series forecasting methods, stating the advantages and disadvantages of both, and briefly describes how probabilistic forecasts can be obtained using these models.

The field of time series forecasting has been dominated by traditional statistical models such as the autoregressive model (AR), the moving average (MA) model, and their many extensions (Adhikari & Agrawal, 2013). This can be attributed to their solid theoretical background, resulting in their properties and behavior being well understood. However, some of these classical methods have limitations when it comes to predicting the more complex time series, as they often rely on assumptions of linear relationships and fixed temporal dependencies (Brownlee, 2018). The application of machine learning methods, including neural networks, has been proposed to overcome some of these limitations.

6.5.1 Statistical Models

All the statistical models presented below assume that the future values of a time series are *linearly* dependent on its past historical observations, and that the data follows a particular known statistical distribution, such as the normal distribution (Adhikari & Agrawal, 2013). These assumptions result in the models being easy to understand, interpret, and develop, explaining why they are widely used for time series forecasting. However, despite these appealing properties, the approximation of non-linear responses using linear models generally does not produce satisfactory results for real-word forecasting tasks (G. P. Zhang, 2001). To implement the models, the optimal model orders must be identified, a task that requires a certain amount of skill and expertise (Box et al., 2015). In the early years, there was no fixed procedure for finding the optimal choice of model, and the model selection was based on the experience of the user. Since then, several techniques for identifying the optimal model has been developed, e.g. Akaike's information criterion (AIC), Akaike's final prediction error (FPE), and the Bayes information criterion (BIC) (De Gooijer & Hyndman, 2006), all aiming to find the optimal model that minimizes the one-step-ahead forecasting errors.

6.5.1.1 Autoregressive Model

An autoregressive model specifies that the value of a time series at time t, $\boldsymbol{z}_t$, is linearly dependent on a specified number p of past values of that time series. For univariate time series is the autoregressive model termed AR(p), and its multivariate extension is termed vector autoregressive model, VAR(p). A vector autoregressive model of order p can be expressed in the following way (Tsay, 2014):

$$\boldsymbol{z}_t = \boldsymbol{c} + \boldsymbol{a}_t + \sum_{j=1}^{p} \boldsymbol{\phi}_j \boldsymbol{z}_{t-j} \tag{4}$$

where $\boldsymbol{\phi}_j$ are the model parameters, $\mathbf{c}$ is a constant vector, and $\boldsymbol{a}_t$ is a stochastic error vector with zero mean vector and positive-definite covariance matrix, whose realizations are independent and identically distributed.

6.5.1.2 Moving Average Model

Unlike autoregressive models, the moving average model, MA(q), uses past forecasting errors, $\boldsymbol{a}_t$, to predict the value of a time series at time t. A vector moving average model of order q, VMA(q), can be expressed in the following way (Tsay, 2014):

$$\boldsymbol{z}_t = \boldsymbol{\mu} + \sum_{i=0}^{q} \boldsymbol{\theta}_i \boldsymbol{a}_{t-i} \tag{5}$$

where $\boldsymbol{\theta}_i$ are the model parameters, $\boldsymbol{\mu}$ is the expectation of $\boldsymbol{z}_t$, and $\boldsymbol{a}_{t-i}$ are the past error vector terms. Equation (4) shows that the value $\boldsymbol{z}_t$ can be considered as the weighted moving average of the past forecasting errors.

6.5.1.3 Autoregressive Integrated Moving Average Model

Autoregressive moving average models, denoted ARMA(p,q), are often used for time series forecasting, and are constructed by combining autoregressive and moving average models, making the forecasts constructed by these models a linear combination of past values of the time series and past errors. The ARMA(p,q) model can only be applied on time series that are stationary[3]. Time series that exhibit trends[4] or seasonality[5] are intrinsically non-stationary, and for such time series extensions of the ARMA processes can be applied, termed autoregressive integrated moving average, ARIMA(p,d,q), and seasonal autoregressive integrated moving average, SARIMA$(p,d,q)\times(P,D,Q,m)$. In these models, non-stationary time series are made stationary by differencing, where the term d indicates the degree of differencing and corresponds to the integrated part of the model. The effect of differencing, using $d = 0, 1, 2$, is shown in the example below:

$$\begin{aligned} z'_t &= z_t, & d &= 0 \\ z'_t &= z_t - z_{t-1}, & d &= 1 \\ z'_t &= z_t - 2z_t{-}1 + z_{t-2}, & d &= 2 \end{aligned}$$

The p, d, q and P, D, Q parameters of the model refer to the trend (lower case) and seasonal (upper case) autoregressive, difference, and moving average order, respectively. Additionally, there exists a fourth seasonal element, m, which influences the other seasonal elements (P, D, Q). For example, for monthly data $m = 12$ suggests a yearly seasonal cycle, and for $P = 1$ the seasonal offset in the model would be $t - (m \times 1) = t - 12$. The order parameters of the model can be zero, which indicates to not include the corresponding element of the model (Brownlee, 2018), e.g. an ARIMA(1,0,0) model reduces to a simple AR(1) model. Additionally, there exist further extensions of the ARIMA models, termed ARIMAX, where the models are expanded, using a linear combination, by including exogenous variables (X).

To identify the optimal model orders, the autocorrelation function (ACF) and partial autocorrelation function (PACF) are plotted to investigate the stationary of the time series. Statistical stationary tests, such as the Dickey-Fuller test (Dickey & Fuller, 1979), are often performed to select the best model. If the ACF and PACF plots show clear correlations, indicating non-stationarity, the time series must be differenced to remove possible trends for the models that require stationarity. After assuring that the time series is made stationary, the AR and MA orders can be identified using the ACF and PACF plots, where p and q are chosen to be as small as possible, within an acceptable error level.

[3]A time series is defined as stationary if all statistical properties of the time series are constant, i.e. do not change with time. For a Gaussian stochastic process this means that the mean and covariance must be constant.

[4]A increasing or decreasing behavior over time

[5]A repeating or cyclic pattern over time

6.5.1.4 Probabilistic Forecasting

Probabilistic predictions, in the form of intervals, can be obtained from ARIMA models using the model residuals, where the intervals are on the general form (Hyndman & Athanasopoulos, 2018)

$$\hat{y}_{t+h} \pm c_\alpha \hat{\sigma}_h,$$

where $\hat{y}_{t+h}$ is the point prediction at time h, c_α is a constant whose value is chosen to get the desired degree of confidence (Shumway & Stoffer, 2017), and $\hat{\sigma}_h$ is an estimate of the standard deviation of the forecast distribution at time h. The estimate of the forecasting distribution's standard deviation is obtained from the standard deviation of the model residuals, which are assumed to be uncorrelated and normally distributed (a strong distributional assumption, yet conventional in autoregressive models, as stated in section 6.5.1). If this assumption is not met, the intervals become unreliable.

As mentioned in section 6.4, a common characteristic of prediction intervals is that as the forecasting horizon increases, the intervals' width grows due to the additional associated uncertainty. (Hyndman & Athanasopoulos, 2018) remark that, in general, for models with $d = 0$, i.e. stationary models, the width of the intervals converges, resulting in that prediction intervals for longer horizons are essentially the same. For models with $d \geq 1$, the prediction interval's width continue to grow in the future.

For one-step ahead predictions, the standard deviation of the residuals is close to the standard deviation of the distribution, and a one-step ahead 95% prediction interval is given by $\hat{y}_{t+1} \pm 1.96\hat{\sigma}$ for all ARIMA models, regardless of model orders (Hyndman & Athanasopoulos, 2018). For other confidence levels, the value of c_α can be found from the standard normal distribution table. For multi-step ahead predictions, the closeness between the standard deviation of the residuals and distribution is not necessarily true, as $\hat{\sigma}_h$ often increases with h and the complexity of the calculations increases. The details of the calculations are beyond the scope of this thesis; for more information, the reader is referred to the book by (Brockwell, Brockwell, Davis, & Davis, 2016).

6.5.2 Neural Network-based Models

The use of machine learning methods, especially neural networks, has been proposed to the problem of time series forecasting with great success, due to their inherent non-linear structure and data-driven approach (Makridakis, Spiliotis, & Assimakopoulos, 2018). Neural networks, explained in-depth in Section 7.1, have the advantage of automatic learning and handling of temporal dependencies and structures such as trends and seasonality (Brownlee, 2018), and support both multiple inputs and outputs. Many time series forecasting problems include large amounts of data, several exogenous variables, and it is often desired to assess multiple related time series to learn patterns within and between the related series. It is in these types of forecasting problems that neural networks have proven to be highly effective at handling such complex relationships.

Some advantages of using neural networks in the field of time series forecasting, compared to traditional statistical methods, are their ability to learn non-linear relationships, the reduced need for significant manual feature engineering, and the fact that they do not require the specification of any statistical distributions (Gasthaus et al., 2019). Since neural networks make no assumptions of the underlying distribution of the data, the forecasts constructed by neural networks are solely based on the information extracted and learned from the data itself.

There are numerous different variations of the traditional feed-forward fully-connected neural network, and some architectures are specifically designed to process a specific type of data. For time series forecasting, where the goal is to forecast a sequence of future values given a sequence of historical observations, it is natural to use network types specifically developed for processing sequential data, such as recurrent or convolutional network types. Recurrent neural networks have in recent years been regarded as the state-of-the-art method for sequence modeling, due to their ability to model temporal dependencies in the input data. Based on these properties, their usage within the field of time series forecasting has proven greatly successful (Chen et al., 2020). RNNs, presented in section 7.3, maintain a memory of all previous input by exploiting recurrent loops within the network's hidden layers. The memory is represented by the network's internal state from the previous time steps, thus providing information about the time series' entire history using a significantly reduced amount of components. This compact representation is particularly beneficial in terms of memory use during training.

Due to their non-linear nature and expressive power can neural networks be termed universal approximators (Hornik, Stinchcombe, & White, 1989), being able to approximate any arbitrary function. Neural networks have in a variety of situations shown to outperform traditional statistical methods, but the results are not exclusive. Real-world data is known to contain noise and outliers, and the amount of available data can be limited. The training of neural networks require a significant amount of data, and neural networks are known to, in some situations, adapt too much to the data, resulting in overfitting, and therefore poor generalization. Network overfitting and methods to reduce its occurrence are explained in section 7.1.3. The need for formal statistical modeling and training is alleviated when working with neural networks, due to their self-adapting nature, and less preprocessing and feature engineering is required compared to other methods, which is advantageous. However, designing a network to perform a specific task can be challenging; choosing the proper network architecture and finding the suitable hyperparameters requires lot of expertise, and can be a time consuming process.

6.5.2.1 Probabilistic Forecasting

As described in Section 6.4, probabilistic predictions can be constructed either directly, using techniques developed with the probabilistic forecasting feature, or via point forecasts. Quantile regression, presented in section 8, is a straightforward, low cost, and fairly undemanding direct method for constructing probabilistic predictions in conjunction with neural networks; any forecasting framework using neural networks can be made probabilistic by simply modifying the loss function to minimize the pinball loss. General loss functions and the pinball loss in particular are described and discussed in section 7.1.2 and 8, respectively.

Alternatively, point forecasts can be extended into probabilistic forecasts by post-processing, or *on-top* frameworks (Hong & Fan, 2016). Post-processing to produce probabilistic forecasting can be done by either utilizing the residuals of the point forecast, or through the combination of several point forecasts. An on-top framework refers to any algorithm that can be placed on top of other algorithms to extend the results. The extension can be performed on point forecasts, making them probabilistic, or on already constructed probabilistic forecasts to improve the existing predictions. Conformal prediction is an example of such on-top frameworks, and is discussed in Section 9.

7 Machine Learning

7.1 Artificial Neural Networks

Artificial neural networks, or neural networks for short, are architectures consisting of connected layers containing computing elements called artificial neurons. The most common neural networks are based on the idea of the *perceptron*, introduced in the 1950s by (F. Rosenblatt, 1958). The perceptron is a simple mathematical model inspired by the workings of the neurons in the human brain, originally formulated as a binary classifier. The perceptron maps the input, $\boldsymbol{x}$, to an output value, $f(\boldsymbol{x})$, according to the following equation:

$$f(\boldsymbol{x}) = \begin{cases} 1 & \text{if} \quad \boldsymbol{\omega}^T\boldsymbol{x} + \omega_0 > 0, \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

where $\boldsymbol{\omega}$, and ω_0 represent the model parameters. The function $f(\boldsymbol{x})$ is termed the threshodling function, or binary step function, and determines the output of the perceptron. The equation above describes the perceptron as a linear model, constructing a single straight line. If the thresholding function is replaced with a nonlinear function, the perceptron becomes a nonlinear model, but a single perceptron is still a linear classifier, as it can only solve linearly separable classification problems. In many cases, a single line is insufficient, e.g., in nonlinear classification or regression problems, and several perceptions can then be utilized together, either placed in parallel, constituting a layer, or in sequence. A network consisting of two or more perceptrons placed in sequence is called a *multilayer perceptron* (MLP). The individual layers of the MLP can contain one or more perceptrons, and are typically fully-connected, where each perceptron in the the current layer is connected to all perceptrons in both the preceding and following layer, then termed a fully-connected layer.

Neural networks can be used to solve a variety of tasks, such as image processing, pattern recognition, classification, time series forecasting, and clustering problems (Theodoridis & Koutroumbas, 2009). The strength of these networks come from their ability to model complex non-linear relationships and learning directly from raw data. The objective of a neural network is to approximate a function, f, that describes the relationship between the input $\mathbf{x}$ and the output $\mathbf{y}$. The mapping from input to output using the function f can be described by the following equation: $\boldsymbol{y} = f(\boldsymbol{x}; \theta)$ (Goodfellow et al., 2016). The network attempts to find the parameters, θ, that produce the best approximation of the true relationship between input and output.

7.1.1 Network Layers

Neural networks consist of a collection of neurons connected in successive layers that together form the network, as described above. There are essentially three different layers in a neural network; the input layer, hidden layers, and the output layer, and the total number of layers in the network is defined as the network depth. The hidden layers are the layers between the input and output layer, and the number of hidden layers included in a network is chosen to best solve the problem at hand. Each layer in the network contains a specified number of neurons, where the maximum number of neurons contained within a layer is referred to as the width of the network.

In a fully-connected feedforward neural network all neurons in the network layers are interconnected, and the output from one neuron therefore affects the output of all subsequent neurons. Figure 1 illustrates a fully-connected feedforward neural network, as well as the workings of a single artificial neuron. The computation taking place at neuron i in a particular layer, l, in a layered network is on the following form (Gonzalez & Woods, 2018):

$$z_i(l) = \sum_{j=1}^{n_{l-1}} w_{ij}(l) \cdot a_j(l-1) + b_i(l) \tag{7}$$

where $z_i(l)$ denotes the input to the activation function of neuron, i, termed the action potential, a_j are the outputs from the activation functions from all neurons in layer $l-1$, w and b denote the weight and bias of the neuron. The weight w_{ij} is the weight of the link between the output of neuron j, in layer l-1, to the input of neuron i in layer l. The output of neuron i in layer l is given by

$$a_i(l) = h\left(z_i(l)\right) \tag{8}$$

where h represents the activation function. There are numerous different activation functions that can be used in neural networks, and the choice of activation function is dependent on the data format of the input to the activation function, the format of the desired output, as well as the task the network is to execute. The role of the activation function is to force the output of the affine transformation taking place within a neuron to lie within a specific range of values. This is done by applying a mapping from input to output, where the mapping is determined by the shape of the activation function.

The activation function determines the output of a neuron, and therefore the level of neuron activation/response. If the activation function is a thresholding function, i.e. the binary step function as in the perceptron, the output of a neuron will be either -1 or 1. If the output of a neuron is 1, then that neuron is said to be activated, hence the name *activation* function. If the output equals -1, the neuron is not activated, and will therefore not contribute to the output of the neurons in the next layer. The drawback of using a activation function with such a hard cutoff is that the function becomes very sensitive, and does not account for the magnitude of the input, only the sign. Additionally, such activation functions are not continuously differentiable, as required by the backpropagation algorithm explained in section 7.1.2. To avoid the problems that result from using the thresholding function, smoother activation functions are applied, where some of the most typically used activation functions include the Sigmoid, hyperbolic tangent (tanh), and Rectifier Linear Unit (ReLU) functions (Gonzalez & Woods, 2018), defined in the equations below. The use of non-linear activation functions is what allows neural networks to approximate non-linear relationships between input and output.

$$\text{Sigmoid} : h(z) = \frac{1}{1+e^{-z}}, \quad h'(z) = h(z)(1-h(z)) \tag{9}$$

$$\tanh : h(z) = \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}, \quad h'(z) = 1-h(z)^2 \tag{10}$$

$$\text{ReLu} : h(z) = max(0,z), \quad h'(z) = \begin{cases} 0 & \text{for} \quad x < 0, \\ 1 & \text{for} \quad x \geq 0 \end{cases} \tag{11}$$

Equation (7) shows that the weighted sum of all input to all neurons in each network is calculated, and a bias term is added. These calculations correspond to a linear operation, where the role of the bias terms in a neural network is to shift the input of the activation function to better fit the estimated outputs, and is analogous to the role of the constant in a linear function. The activation function is applied to the output of summation operation in equation (7), and the output from the activation function is the output of the neuron. The nodes in Fig. 1 represent the network neurons, and the lines between the nodes represent the weighted connections between each neuron. In this network, all lines are carrying information forward in the network and the information flows strictly forward through each node from the input to the output layer, as required in a fully-connected feedforward neural network. There exist several extensions of the MLP, and different network types have different distinct layers, each designed to perform specific tasks.

After choosing an appropriate network type for the problem at hand, there are several other network design parameters that must be determined; the number of neurons in the input, hidden, and output layer, as well as the number of hidden layers. The number of neurons in the input layer is dependent upon the format of the input data. For tasks such as time series forecasting, the number of neurons in the input and output layer corresponds to the number of historical observations included in the forecasting model, and the number of future time steps that are to be predicted, respectively. Choosing the number of hidden layers and neurons within them is not as straight-forward as for the first two; there are no general rules, and the choice depends upon the use case. Generally, the number of hidden layers one should include is related to the complexity of the problem the network is set to solve. If the data is linearly separable, there is no need for hidden layers, but as the complexity of the data increases, so should the number of hidden layers. This is only the case up to a particular point, where it has been shown that too deep networks have a decreasing performance compared to shallower alternatives (He, Zhang, Ren, & Sun, 2016). A discussion of and a solution to this problem is described in section 7.2.

Hyperparameters such as the number of hidden layers and neurons per layer are in practice selected by evaluating the model performance on the validation dataset using hyperparameter optimization techniques such a manual search, grid search or random search. These methods randomly or systematically selects hyperparameters from a user-defined search space; a volume where each dimension represents a hyperparameter, and each point within represents a particular model configuration (Brownlee, 2018). Randomly selecting and evaluating hyperparameters within the search space is termed a random search, whereas searching the space systematically by evaluating hyperparameter configurations

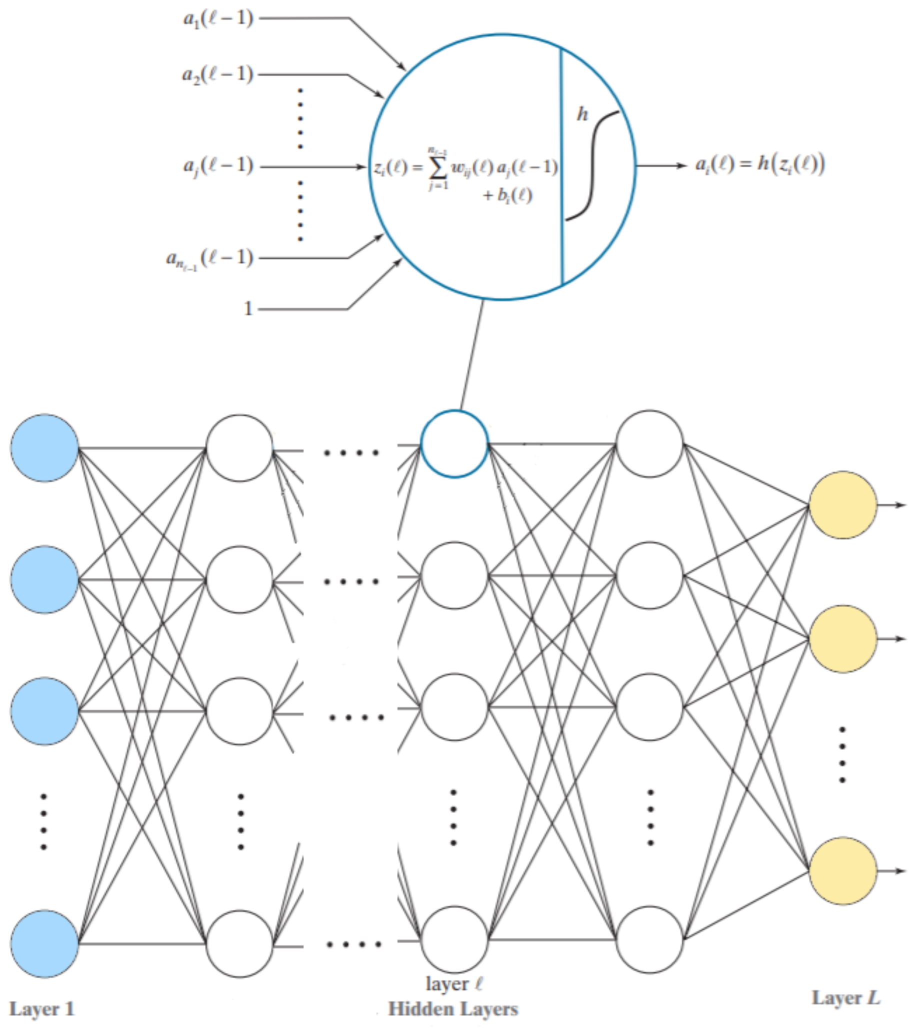


Figure 1: Fully-connected feedforward neural network. The figure inspired by Fig. 12.31 in (Gonzalez & Woods, 2018).

sampled within the space, is referred to as a grid search. The search can be performed using a fixed training and validation dataset pair, or k-fold cross-validation. The latter is often used if the original dataset is small, where the division into training, validation and test datasets often results in the test set being small. The cross-validation technique avoids this problem by repeating the training and validation computation on k different randomly non-overlapping subsets of the original dataset (Goodfellow et al., 2016), reporting results averaged over all folds.

7.1.2 Network Training

A neural network is entirely described by its parameters, i.e. the weights and biases, and activation functions of each layer in the network. Network training is the iterative process of finding the optimal values for the randomly initialized trainable network parameters. The optimal parameter values are the values that maximize network performance, producing the most accurate approximation of the mapping from input to output. A neural network learns by solving a task, measuring the model performance on a set of training samples, and updating the model parameters accordingly. The performance measure is quantitative and depends on the task at hand. Learning is, as stated in section 2, often divided into two main paradigms; supervised and unsupervised learning. Supervised learning algorithms use a collection of labeled samples, with the goal of finding the parameters that best predict the true output, i.e. the labels. In unsupervised learning the samples have no labels, and the objective is to uncover and learn the underlying structures of the sample data.

In a supervised learning problem the data is commonly split into three separate data sets; training data, validation data and test data. The model is trained using the training data, model hyper-parameters are chosen using the validation data, and finally, the model performance is evaluated using the test data. The reason for using two separate data sets to train and evaluate the model is to assess the generalization capability of the model.

The training of neural network is divided into four main steps:

1. First, the network parameters are initialized with random values. The initialization of the network's weights and biases represents the starting point of the iterative network optimization procedure, and determines how quickly the network converges and to what value of the cost function it converges. The initial network weights are often drawn randomly from a Gaussian or uniform distribution (Goodfellow et al., 2016), whereas the biases are commonly initialized to zero. The scale of the initial distribution affects both the results of the optimization procedure and the network's ability to generalize.

2. Next, the training samples are propagated through each layer in the network, an output is computed, and the error measurement between the estimated and true (or desired) output is computed, i.e. the value of some loss function for the current parameters is calculated. This second step is called a forward pass or forward propagation through the network. The loss function is a measurement used to evaluate the candidate network parameters, and must be constructed in a way such that it encapsulates the problem the network is set to solve. For neural networks, the problem most often involves error minimization, and in supervised learning problems the loss function, denoted $L(\boldsymbol{\theta})$, portrays the error between the estimated output and the true output. The optimal set of network parameters is the set that minimizes the total network loss, i.e. the summation of the error obtained for each sample in the training data. A commonly used loss function is the *mean squared error* (MSE), defined as the average of the squared error between the output and the labels:
$$L(y, \hat{y}; \boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^{N} (\hat{\boldsymbol{y}}_i(\boldsymbol{\theta}) - \boldsymbol{y}_i)^2 \tag{12}$$

3. After the scalar value of the loss function is found, a way of estimating the necessary parameter updates is required. The backpropagation algorithm, introduced by Rumelhart, Hinton, and Williams in 1986 (Gonzalez & Woods, 2018), is a gradient-based algorithm that provides an efficient way of computing the required parameter

updates. The backpropagation algorithm propagates the obtained error measurement backward through the network, finding the required parameter changes by calculating the gradient of the loss. This third step is referred to as backpropagation through the network.

4. The last and fourth step is to update the network parameters according to the result obtained from the first three steps, where the gradient value is used to adjust the network parameters. The degree to which the parameters change is dependent on the value of the gradient; the bigger the gradient, the bigger the adjustment. In the iterative gradient descent algorithm, presented in the following subsection, the parameter update is also dependent on a user-defined step size, termed learning rate. Step two and three of the training process are repeated a specified number of times, or until the parameters stop updating, where one iteration through these steps is called an epoch.

7.1.2.1 Network Optimization Algorithms

Optimization is defined as the task of maximizing or minimizing a function $f(\boldsymbol{x})$ by altering $\mathbf{x}$ (Goodfellow et al., 2016). Gradient-based optimization algorithms seek to find the local minimum of the loss function by moving in the negative direction of the gradient found during the backpropagation step of the network training. The most commonly applied optimization algorithm for network training it the stochastic gradient descent (SGD), along with its many variants (Goodfellow et al., 2016). The SGD algorithm uses the derivative of the loss function to find the optimal parameters. The derivative of a function at a given point gives the slope of the function at that point, and utilizing this property of the gradient is practical because it explicitly expresses how to change the parameters in a way that minimizes the loss function. This SGD algorithm states that changing the parameters in a way that corresponds to moving in small steps in the negative direction of the gradient will result in finding the minimum of the function. The size of the step is termed as the learning rate, a hyperparameter that needs to be specified before training. The trainable network parameters, $\boldsymbol{\theta}$, are optimized by updating them as following

$$\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i - \eta \nabla_{\boldsymbol{\theta_i}} L(\boldsymbol{\theta}_i) \tag{13}$$

where η denotes the learning rate. The optimal value of the learning rate is both dependent on the model used, and on the data presented to the model. There are therefore no clear rules for how to choose the value of the learning rate. It is most often found by trial and error, which can make the tuning of this hyperparameter challenging.

The SGD method estimates the gradient from randomly selected subsets of the training dataset, called mini-batches. If the gradient of the entire training dataset is used, then the method is no longer stochastic and reduces to the standard gradient descent method. One of the reasons for using mini-batches instead of the whole dataset, is that if the amount of data is huge, which for neural networks is desirable, the computational cost of calculating the gradient can be intolerably large. The SGD algorithm therefore updates the network parameters by calculating the gradient on each mini-batch, and updating the parameters accordingly. Additionally, updating the network parameters using mini-batches introduces some noise in the learning process, helping the network to avoid getting stuck in local minima. One drawback of estimating the true gradient by the gradients of mini-batches is that it converges slowly, and extensions of this algorithm have been presented for improvement.

Momentum can be included in SGD to help the convergence rate. The momentum term is an exponentially decaying average of the previous gradients (Goodfellow et al., 2016), and when including momentum, the network parameters are updated according to a linear combination of the currently estimated gradient, and the previous update (Bianchi, Maiorino, Kampffmeyer, Rizzi, & Jenssen, 2017). Momentum assumes that the previous update was an update in the right direction and that one should therefore continue to move in that direction. This assumption builds up velocity towards the direction that shows a consistent gradient, which reduces the number of oscillations between multiple directions, thus accelerating the convergence.

The Adaptive Moment Estimation Optimization Algorithm, or Adam for short, is an extension of the momentum stochastic gradient descent algorithm where separate adaptive learning rates are computed for each network parameter from estimates of first and second moments of the gradients (Kingma & Ba, 2014). The Adam algorithm stores an exponentially decaying average of the squared gradients and an exponentially decaying average of the moments of the gradients, and uses both to find the new parameter update. By including the first moment, i.e. the mean gradient, the parameter update is improved at points where the gradient is small. The variance, which is a second-order moment, is included to ensure that if the variance is large, and therefore the uncertainty is high, the learning rate is reduced to only take small steps in the direction of the descending gradient.

7.1.3 Network Regularization

One common, unwanted problem that occurs when training a neural network is that the network can learn and adapt too well to the training data. This problem is referred to as overfitting and results in high training accuracy, but poor test accuracy. An overfitted network generalizes badly, and will therefore perform poorly on unseen data. The number of network parameters naturally increases with increasing network size, often resulting in neural networks having hundreds, thousands, or in some cases millions of parameters. If the amount of data used to train a neural network is much less compared to the number of network parameters, the network will be prone to overfitting. Having large sets of training data therefore reduces the chance for overfitting and improving network generalization. Further, several methods can be applied to reduce the effect of overfitting, where three commonly used methods are dropout, batch normalization, and L2 regulation, all explained below.

7.1.3.1 Dropout

Dropout is a network regularization technique where a number of randomly selected network nodes are excluded during training, i.e. the incoming and outgoing connections of a node are *cut off*. Dropout can be included in a network by adding dropout layers during network training, where the probability of a node to be dropped, often referred to as the dropout rate, is a hyperparameter that must be determined before training, where the optimal dropout rate is commonly found via a hyperparameter search during validation (Goodfellow et al., 2016).

7.1.3.2 Batch Normalization

During the training of a neural network the parameters of the network layers are repeatedly updated in order to obtain the optimal set of network parameters. The process of network training is complicated by the fact that the inputs to each network layer are affected by the parameter change of the preceding layers (Ioffe & Szegedy, 2015). This change in the distribution of the layer inputs complicates the training, because it leads to the network layers continuously needing to adapt to new distributions of the inputs. This effect is called the *internal covariate shift* and a solution to this problem is batch normalization. Batch normalization re-centers and re-scales the distribution of the layer inputs for each mini-batch, making it have zero mean and variance of 1, which can help improve network performance and reduce training time significantly.

7.1.3.3 L2 Regularization

The L2 regularization technique reduces the occurrence of network overfitting by limiting the possible values of the network's weights, leaving the biases unregularized. The reason for only applying regularization on the network weights is that the biases are often easier to fit accurately, since they only depend on one variable, whereas the weights specify how two variables interact (Goodfellow et al., 2016). The network weights are determined during network training, and are real-valued, either positive or negative. Overfitted networks are often characterized by having large weights, and the L^2 regularization technique restricts the scale of the weight values, not allowing them to become too large.

The regularization term is added to the network loss function, creating a new regulated loss function:

$$L = L(\boldsymbol{\theta}) + R_\lambda(\boldsymbol{W}),$$

where R_λ is a regularization function and λ is a hyperparameter that weights the contribution of the regularization in the total loss (Bianchi et al., 2017). For L2 regularization, R_λ is defined as

$$R_\lambda(\boldsymbol{W}) = \lambda_2||\boldsymbol{W}||_2,$$

penalizing network parameters with large magnitudes. If λ_2 is very small or zero, the total loss reduces to the original unregulated loss function, resulting in no regularization. If λ_2 is very large, it will add a substantial contribution to the total loss, corresponding to more regularization, which can lead to underfitting. Finding the optimal value of λ_2 is therefore essential.

L2 regularization can be applied to the weights in specific network layers or all of the network's weights collectively. Consequentially, separate penalty coefficients, $lambda_2$, can be used for each layer in the network, but this requires an extensive search for the optimal coefficient of each layer, and a single coefficient is often used for all layers to reduce the size of the search space (Goodfellow et al., 2016).

7.2 Residual Neural Networks

Residual networks were first introduced in 2015 by (He et al., 2016) in the paper *Deep Residual Learning for Image Recognition.* The building blocks of these networks are named residual blocks, which are introduced to overcome the difficulty associated with the training of very deep networks. This difficulty is referred to as the degradation problem, a very counter-intuitive problem that occurs during training of very deep networks, where deeper networks are shown to produce lower training accuracy than shallower nets. The problem follows from the fact that as networks becomes very deep, the propagation of information from the earliest layers becomes more challenging, and the information is lost. Note that this is not the problem that occurs when a model is overfitted, i.e. the model learns the training data too well, which causes the *test accuracy* to decease. The degradation problem is counter-intuitive, because one would expect the *training accuracy* to improve when the number of layers is increased due to the increased expressive power. Instead, when networks become very deep, the training accuracy starts to decrease, causing the deeper networks to perform worse compared to their shallower counterparts.

Residual networks solve the degradation problem by introducing residual blocks containing skip connections that allow information to flow across and by-pass one or more layers without being affected by the non-linearity of the intermediate layers. The residual connection is an element-wise addition between the input and latent output of the residual block, illustrated in Fig. 2. The hidden layers within the residual block can be any type of network layers, e.g. convolutional or fully-connected layers.

The residual connections are constructed such that the layers learn residual functions with reference to the layer inputs (He et al., 2016). This means that instead of trying to model the true mapping relationship from input to output, the layers are constructed to fit a residual mapping $F(\boldsymbol{x}) = H(\boldsymbol{x}) - \boldsymbol{x}$, where $H(\boldsymbol{x})$ denotes the original mapping. These residual connections force the network to approximate the residual function $F(\boldsymbol{x})$ instead of the true mapping $H(\boldsymbol{x})$, and these layers can therefore learn identity mapping by setting the weights equal to zero.

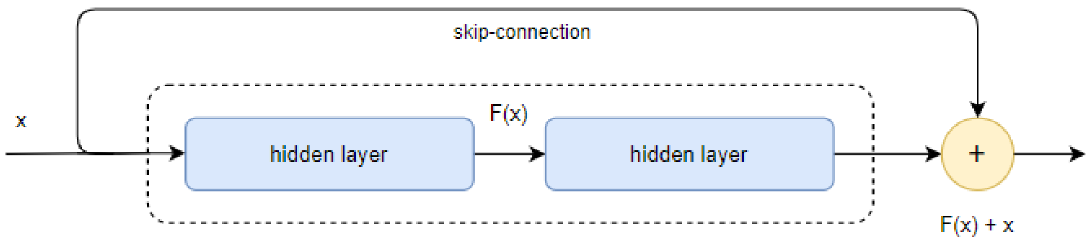


Figure 2: Residual block containing two hidden layers. The ⊕ represent an element-wise addition.

7.3 Recurrent Neural Networks

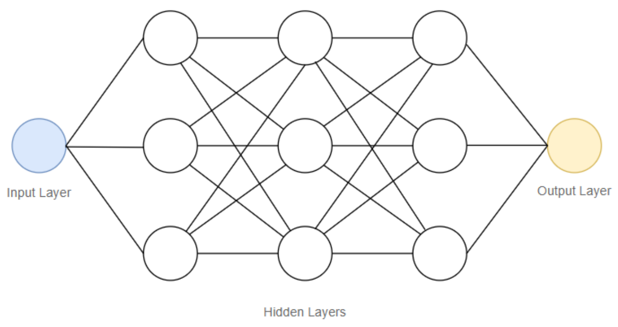
Recurrent neural networks is a class of neural networks categorized by internal recurrent connections, used for processing sequential data (Goodfellow et al., 2016). Unlike feedforward neural networks, where information only flows in one direction through the network, the use of the recurrent connections allows the information to flow in loops within the network. The computations taking place at time step t in a RNN depends not only on the input at that time, but also on the preceding time steps. The motivation for including internal connections is that sequential data, e.g. time series, contain important temporal dependencies and repeating patterns which can be difficult to exploit in ordinary feedforward networks; These networks have no memory of the previous inputs, and constructs predictions based on a user-defined fixed window of previous inputs. This method may work well in some cases, but it can be difficult to find the optimal window size. Additionally, for some data the optimal window size can vary dynamically. Hence, constructing networks that process data in a sequential manner allows for the natural temporal dependencies within the whole of a sequence to be learned and exploited.

The internal recurrent connections in a RNN can be interpreted as the memory of the network, contained in the hidden layers of the network. In the same manner as the computations described in Eq. (7), RNNs assign weights and biases to the input at time t, but also apply a weighting to the recurrent connections. After applying weights and adding biases to the input and previous internal state, the two are added together and an activation function is applied, creating the new internal state. The output of the network is computed by an often linear transformation of the network's internal state and an added bias term. This can be explained by the following equations (Bianchi et al., 2017):

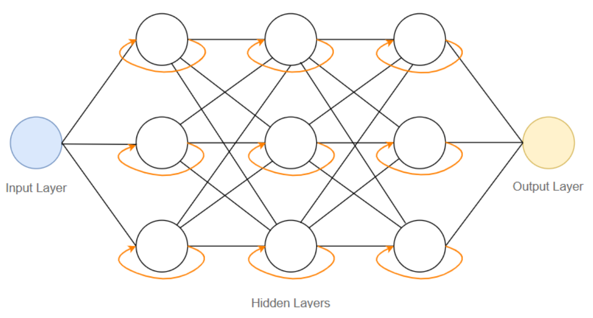
$$\begin{aligned} \boldsymbol{h}[t] &= f\left(\boldsymbol{W}_i^h(\boldsymbol{x}[t] + \boldsymbol{b}_i) + \boldsymbol{W}_h^h(\boldsymbol{h}[t-1] + \boldsymbol{b}_h)\right) \\ \boldsymbol{y}[t] &= g\left(\boldsymbol{W}_h^o(\boldsymbol{h}[t] + \boldsymbol{b}_o)\right) \end{aligned} \tag{14}$$

where $\mathbf{x}$, $\mathbf{y}$, and $\mathbf{h}$ represent the input, output and hidden nodes, respectively, $\boldsymbol{W}_i^h, \boldsymbol{W}_o^h$, and $\boldsymbol{W}_h^h$ denote the weight matrices of the input, output and hidden layers, f is the activation function, and $g(\cdot)$ denotes the transformation function used to produce the network output.

Deep recurrent networks can be constructed by stacking several recurrent layers, where each hidden state of the recurrent layers is passed to both the next time step of the current recurrent layer and to the current time step of the next recurrent layer. An illustration of a recurrent neural network, and a comparison with a fully-connected neural network, is shown in the Fig. 3



(a) Fully-connected neural network



(b) Recurrent neural network

Figure 3: Illustration of a recurrent neural network

7.3.1 Vanishing and Exploding Gradients

One drawback of using the simple RNN described above is the *vanishing or exploding gradient problem*; a problem that emerges during the backpropagation part of the network training. For RNNs, the backpropagation algorithm is referred to as backpropagation through time, BPTT, where the network is *unfolded* in time, i.e. replicating the network's internal state for each time step, illustrated in Fig. 4. During BPTT the adjustments at a time t is dependent on adjustments made at the previous times. Hence, if the adjustments at time t are small, the adjustments at the preceding time $t-1$ will be even smaller, causing the gradient to exponentially shrink, i.e. the gradient is vanishing. The shrinking of the gradients results in practically no adjustments being made at the earliest time steps, resulting in little learning at these times. The vanishing gradients cause RNNs to have a short-term memory, only regarding information about times closest to the current time step. If the adjustments during BPTT are very large, the update of the network weights will be correspondingly large, causing the gradient to exponentially increase due to the repeating multiplication of gradients with value greater than 1. The exploding gradients result in the network becoming unstable and unable to learn from the training data. Exploding gradients can be avoided by regularization, e.g. L2 regularization.

To avoid the vanishing gradient problem, extensions of the basic RNN architecture have been introduced; the Long short-term memory (LSTM) and the gated recurrent unit (GRU) network. Both LSTM and GRU networks are based on the principles described above, but avoid the vanishing gradient problem by introducing a memory cell that replaces the hidden neurons in the network. The memory cell represents the networks internal state, and there are no non-linearities between adjacent cell states in time, allowing the cell state to be preserved without being amplified or shrinked over time. Within the cell, gating mechanisms regulating the flow of information are utilized. The gates manipulate the cell state by the addition and removal of relevant and irrelevant information, deciding how much information from the previous time steps that is let through the gate, thereby influencing the current internal state. The LSTM network is explained in the following subsection.

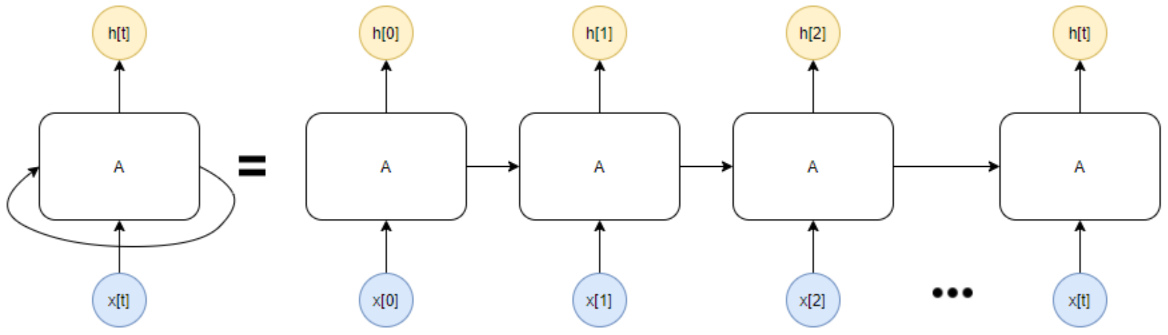


Figure 4: A recurrent neural network, A, unfolded in time

7.3.2 Long Short-term Memory

The long short-term memory architecture is an extension of the simple RNN, explicitly designed to tackle the short-term memory problem, thus making RNNs capable of learning long-term dependencies. As described above, LSTM networks solve the vanishing gradient problem by introducing an additive gradient structure. Following the structure of the simple RNN, LSTM networks contains a chain of repeating components, as in Fig. 4, but the components now have a different structure; Instead of a single non-linearity, as in Eq. (14), an LSTM cell is composed of five different nonlinear components (Bianchi et al., 2017):

$$\begin{aligned} \text{forget gate} : \sigma_f[t] &= \sigma(\boldsymbol{W}_f \boldsymbol{x}[t] + \boldsymbol{R}_f \boldsymbol{y}[t-1] + \boldsymbol{b}_f), \\ \text{candidate gate} : \tilde{\boldsymbol{h}}[t] &= g_1(\boldsymbol{W}_h \boldsymbol{x}[t] + \boldsymbol{R}_h \boldsymbol{y}[t-1] + \boldsymbol{b}_h), \\ \text{update gate} : \sigma_u[t] &= \sigma(\boldsymbol{W}_u \boldsymbol{x}[t] + \boldsymbol{R}_u \boldsymbol{y}[t-1] + \boldsymbol{b}_u), \\ \text{cell state} : \boldsymbol{h}[t] &= \sigma_u[t] \odot \tilde{\boldsymbol{h}[t]} + \sigma_f[t] \odot \boldsymbol{h}[t-1], \\ \text{output gate} : \sigma_o[t] &= \sigma(\boldsymbol{W}_o \boldsymbol{x}[t] + \boldsymbol{R}_o \boldsymbol{y}[t-1] + \boldsymbol{b}_o), \\ \text{output} : \boldsymbol{y}[t] &= \sigma_o[t] \odot g_2(\boldsymbol{h}[t]) \end{aligned} \tag{15}$$

where $\mathbf{x}$[t] and $\mathbf{y}$[t] represent the input and output at time t, respectively, $\boldsymbol{W}_f, \boldsymbol{W}_h, \boldsymbol{W}_u$, and $\boldsymbol{W}_o$ denote rectangular weight matrices applied to the input of the cell, $\boldsymbol{R}_f, \boldsymbol{R}_h, \boldsymbol{R}_u$, and $\boldsymbol{R}_o$ are square matrices representing the weights of the recurrent connections, $\boldsymbol{b}_f, \boldsymbol{b}_h, \boldsymbol{b}_u$, and $\boldsymbol{b}_o$ are bias vectors, $g_1(\cdot)$ and $g_2(\cdot)$ are non-linear activation functions (tanh activation functions are often utilized in LSTM), and $\odot$ denotes element-wise multiplication. The gating mechanisms, σ, are constructed using sigmoid activation functions, mapping the input values to lie in the interval (0,1), where 0 represent a closed gate, letting no information through, and 1 an open gate, letting all information through. The gates in the LSTM cell are in practice never completely open or closed, due to the open interval of the sigmoid function.

The three gating mechanisms in an LSTM cell are the forget-gate, input-gate, and output-gate, each designed uniquely with distinct functionalities. At a time t, the forget-gate, σ_f, controls how much of the previous cell state, $\boldsymbol{h}[t-1]$, that is to be discarded given the current input. The input-gate σ_i decides, based on the output from the forget-gate, how much of the proposed new state, $\tilde{\boldsymbol{h}}[t]$, that should be added to the current cell state, $\boldsymbol{h}[t]$. The last gating mechanism is the output-gate, σ_o, which determines the output, $\boldsymbol{y}[t]$, which is a filtered version of the cell state. An illustration of an LSTM cell and the gating mechanisms are shown in Fig. 5.

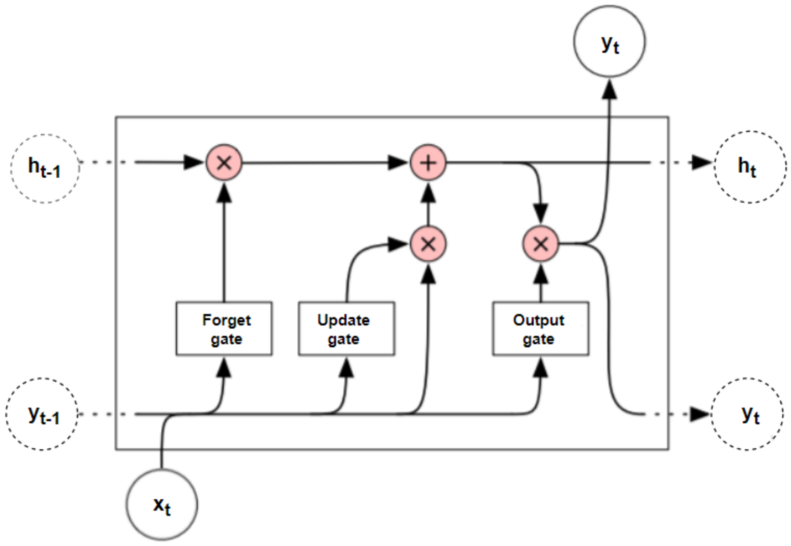


Figure 5: Illustration of a long-short memory cell, inspired by Fig. 3 in (Bianchi et al., 2017). The solid circles represent the variables whose content is exchanged with the input and output of the cell. The dashed circles represent the internal state variables, whose content is exchanged between the cells of the hidden layer.

7.4 Convolutional Neural Networks

A convolutional neural network is a specialized kind of neural network most often used on two-dimensional data, i.e. data presented as arrays or grids, but can be used on one-dimensional data by treating a sequence of observations like a one-dimensional array (Brownlee, 2018). CNNs were first introduced by (LeCun et al., 1989) for classifying images of handwritten digits, but have in recent years been applied to more complex images and other tasks, such as speech recognition (Abdel-Hamid et al., 2014) and time series forecasting (Borovykh et al., 2017). Unlike fully connected neural networks, where every node in the network layer is interconnected, convolutional neural networks are based on the idea of local connectivity, described below.

7.4.1 The Convolution Operator

The name *convolutional* neural network originates from the mathematical operation performed in signal processing, where a filter is slid (i.e., convolved) over an image. Convolution, a linear mathematical operation, inputs two functions to create a new third function, expressing how the shape of one function modifies the other. The convolution between an image, $f(x, y)$, and a kernel $w(x, y)$ with size $m \times n$, can be expressed in the following way (Gonzalez & Woods, 2018):

$$(w * f)(x, y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s, t) f(x - s, y - t) \tag{16}$$

where $a = (m - 1)/2$, $b = (n - 1)/2$, and $*$ denotes the convolution operator. The kernel can be viewed as a sliding filter, extracting important features at different locations in the image, but is essentially just an array containing weights that are optimized during training. The size of the convolutional kernel is made smaller than the size of the image, and the convolution between the kernel and the image pixels covered by the kernel are calculated for each pixel in the image by sliding the kernel across the whole image. The output of the convolutional operation is a single value, which is analog to the output of a neuron in a fully connected neural network.

As a result of the kernel being made smaller than the input image, the size of the output image is reduced compared to the input image's size, due to the loss of information contained in the edges of the input image; The kernel is applied first at the top left of the image, and then systematically moved across the whole image. This results in the edges of the image not being exposed to the center of the kernel, which produces a smaller output image. If the network contains several convolutional layers the size of the output can be significantly reduced, which in some cases is undesirable. To prevent this effect, padding can be applied to the input image. Padding increasing the size of the input by adding a specified number of pixels at the borders of the image, such that the resulting has the same size as the original input. One common padding method is zero-padding, where zero-valued pixels are added as a frame around the input image (Gonzalez & Woods, 2018).

7.4.2 Layers in Convolutional Neural Networks

The most common structure of a convolutional neural network for classification or regression tasks includes convolutional layers, pooling layers, and lastly a fully connected layer. The convolutional and pooling layers are added to extract the important features of the input image, and the output of the last pooling layer is flattened into a single vector and fed into a fully connected layer. The fully connected layer is designed to perform the task at hand: for classification, the output of the fully connected layer predicts which class the input belongs to; for time series forecasting the output is the predicted real-values of a time series.

7.4.2.1 Convolutional Layers

In CNNs, convolutional layers can have several kernels, which together form a 3-dimensional filter. Each kernel produces one output, or feature map, and the number of feature maps created by the convolutional layer is therefore the same as a third dimension of the filter. The operation taking place at a single kernel in the convolutional layer l in a convolutional network can be expressed by the following equation:

$$z(l) = h\left(K(l) * a(l-1) + b(l)\right) \tag{17}$$

where K denotes the convolutional kernel, and the other symbols represent the same entities as in Eq. (7).

In a convolutional layer, the same kernel weights are used for all locations of the input, resulting in the network only having to learn one set of weights for each kernel in the filter. The use of the same parameters at every location in the input enables the network to detect the same type of patters at arbitrary location in the input, an important characteristic of CNNs termed *parameter* or *weight sharing*. The practice of parameter sharing in CNNs makes it possible to apply the model to inputs of different sizes/lengths and generalize across them, detecting features independent of where they are located in the input (Goodfellow et al., 2016). Additionally, sharing the kernel weights across the spatial dimensions of the image reduces the number of network parameters. Compared to a fully-connected neural network where the number of parameters needed to process an image is the same as the number of pixels in the image, the number of network parameters in a CNN is significantly lower, which is desirable in terms of memory consumption during network training, but most importantly, prevents overfitting and allows better generalization.

7.4.2.2 Pooling Layers

The output from the convolutional layer, i.e. the output from the activation function, is pooled or sub-sampled using pooling layers. These pooling layers reduce the volume of the feature space by dividing the feature maps into sets of small adjacent regions, or pooling neighborhoods, and replace all pixels in a region with a single value, producing one pooled feature map for each feature map in the feature space. There exist several pooling methods, where one frequently used method is *max-pooling*. The max-pooling method works by replacing the values within the pooling neighbourhood with the maximum value within the neighbourhood.

The motivation for adding pooling layers in CNNs is not only to reduce data dimensionality, but also because pooling over spatial regions produces invariance to translation, making the network less sensitive to changes in feature positions in the input image (Goodfellow et al., 2016), and increases the receptive field of the network.

7.4.3 Network Receptive Field

The receptive field of a network is the area of input which is available for a specific layer unit (Luo, Li, Urtasun, & Zemel, 2017). For fully connected neural networks, each network unit is dependent on the entire network input, but this is not the case for convolutional networks; In CNNs only a specific, local region, or neighbourhood, of the network input is available for each network unit successively, and the size of that region is referred to as the networks receptive field.

There are several ways to increase the receptive field of a convolutional neural network; using deeper networks, i.e. adding more layers to the model, increasing kernel size, using pooling layers, and using strided or dilated convolutions (discussed in Section 7.5.1), where a combination of these methods are often employed. An illustration of the receptive field of a convolutional network is shown in Fig. 6. Large receptive fields are often desired since they allow large areas of the original input to be visible for each individual neuron in the network.

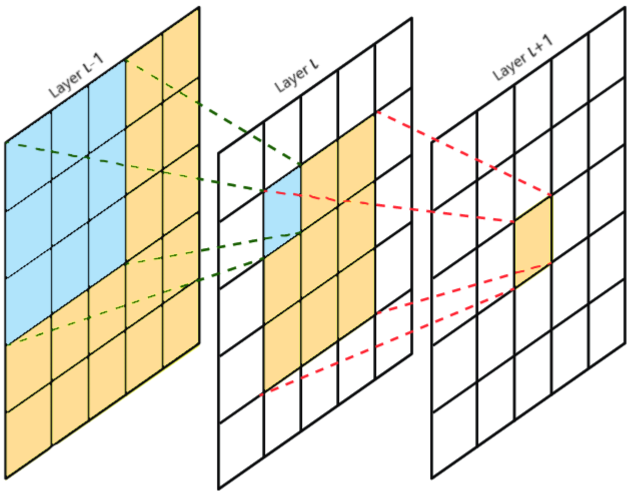


Figure 6: Network receptive field with 3x3 kernel

As the figure above illustrates, only smaller local areas of the whole input are available for the neurons within a convolutional layer, detecting small and simple features, like edges in an image. By stacking several convolutional layers the size of the area available for each neuron grows, and the combined information extracted from several small areas allow the capturing of high level features such as complex object shapes. The joint exploitation of small areas of the input, extracted by adjacent layers is referred to as local connectivity (Gonzalez & Woods, 2018).

7.5 Temporal Convolutional Neural Networks

A temporal convolutional network is a variant of convolutional neural networks, first presented by (Lea et al., 2017), that combines the main concepts of CNNs and RNNs into one model; TCNs are able not only to identify and learn meaningful repeating patterns within the input data, identical to CNNs, but also to capture temporal dependencies within the sequence, and map input sequences to output sequences of equal length, just as RNNs. TCNs are mainly used for sequential data, where the convolutional kernel is designed in such a way that important local patterns within the sequence is extracted and learned. CNNs are, as stated in section 7.4, mostly used for input data in the form of arrays, e.g., image analysis, and the convolutions are therefore spatial. The convolutions in TCNs are in contrast computed across time, oftentimes with a causal constraint and increasing dilation factors, both explained below in Section 7.5.1. There is no implicit requirement for the convolutions in a TCN to be causal, but non-causal convolutions result in TCNs not being suitable for real-time applications.

The receptive field of networks containing convolutional layers increases with dilation factor, kernel size, and network depth, and the use of increasingly dilated causal convolutions results in large receptive fields, without the need for very deep networks and large filters. In time series forecasting, large receptive fields are desired, since only information extracted from observations contained within the network's receptive field are considered when constructing the predictions. To aid network training when the size and depth of TCNs increases, residual blocks are commonly utilized, taking the place of the ordinary convolutional layers, and containing at least one convolutional layer.

The characteristic pooling layers in CNNs are optional in TCNs, since the use of dilated convolutional kernels achieves a similar effect as the pooling operations; Increasing dilation factors result in the convolutional layers operating on a smaller number of time steps sampled at time intervals that increase throughout the layers, ensuring the capturing of important fine details and repetitive patterns, while the number of considered time steps is kept moderately small (Lea et al., 2017). Additionally, increasing the dilation factor with each layer increases the receptive field of the network exponentially, while the number of trainable parameters grows linearly. Dilated convolutions, together with the nature of convolutional layers, i.e. local connectivity and parameter sharing, result in temporal convolutional networks having a relative small number of trainable network parameters, which is favorable.

From the information presented above is it clear that TCNs are well suited for the problem of time series forecasting. However, notable disadvantages exist (Bai, Kolter, & Koltun, 2018). Contrary to RNNs, where each internal state represents a "summary" of the entire preceding history of the input sequence in the form of previous internal states, TCNs have access only to the information contained within their receptive field. For a TCN to include more observations within its receptive field, the size of the window that covers the historical observations must increase either by increasing the kernel size, dilation factor, or number of convolutional layers. The choice of kernel size and dilation factors is generally highly dependent on the length of the input sequence and the variable being predicted, and are additional hyperparameters that must be tuned specifically for the problem at hand.

7.5.1 Dilated Causal Convolutions

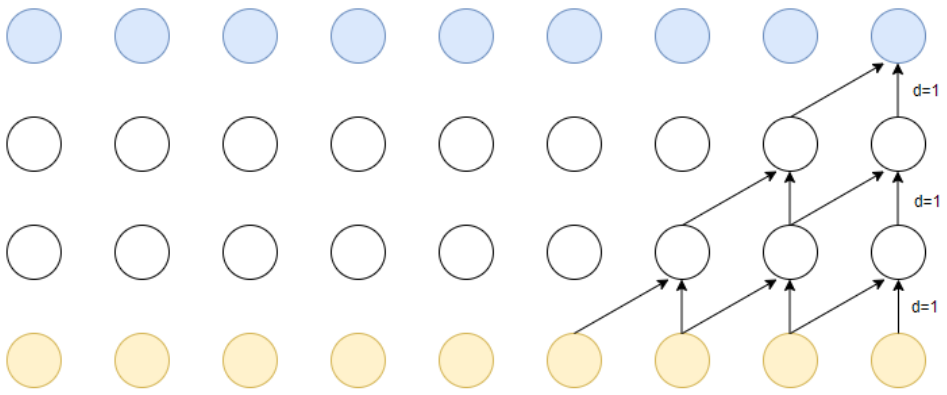
A convolution is said to be causal if the output at the time t only depends on inputs from previous time steps $t-1, t-2, t-3, \ldots$, meaning that no information from future time steps is available when computing the internal representations. Dilated convolutions are convolutions where the kernel is designed in such a way that it has a field of view larger than its actual size. A dilated kernel increases its field of view by ignoring a specified number of steps between each input value. For example, a dilated convolution kernel with size 2 and a dilation factor 4 will only include input values at every fourth step, and is analogous to a kernel with size 5 where the weights for the steps between the first and the last entry are zero. A kernel with dilation factor of 1 is equivalent to an ordinary convolution. Figure 7 illustrates causal and non-causal dilated convolutions. For a univariate time series, the output, $s(t)$, of a causal convolution with a kernel with dilation factor d and kernel size K at time t can be expressed as (Chen et al., 2020):

$$s(t) = (x *_d w)(t) = \sum_{k=0}^{K-1} w(k)x(t - d \cdot k) \tag{18}$$

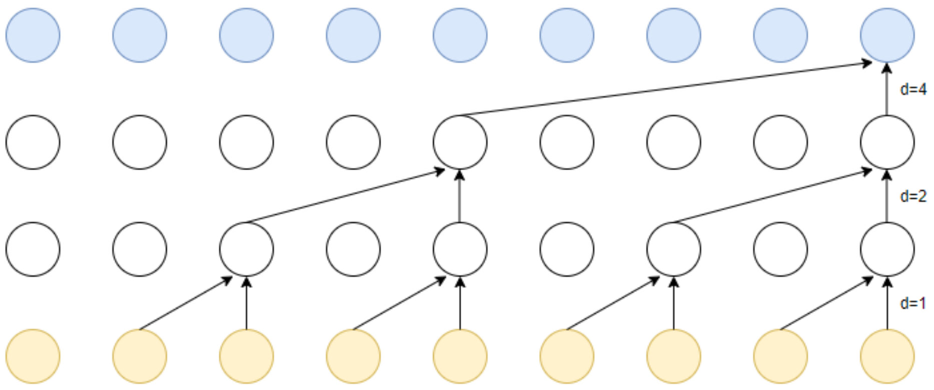
where $*_d$ denotes the dilated convolution with dilation factor d.

Dilated convolutions differ from strided[6] convolutions, since the input resolution is preserved throughout the convolution, meaning that the input and output dimensions are equal. Stacking dilated convolutions significantly increases the receptive field of a network with a small number of layers (Oord et al., 2016), enabling shallow networks to have very large receptive fields while still preserving the input resolution. In time series forecasting, the use of dilated convolutions in the convolutional layers in TCNs reduces the volume of considered input data for each time step, while still including a wide selection of historical observations when predicting future values of the time series. Stacking multiple dilated convolutional layers ensures that only important patterns within the input are extracted, and unwanted noise is discarded. However, some fine details and fast dynamics within the time series might be overlooked.

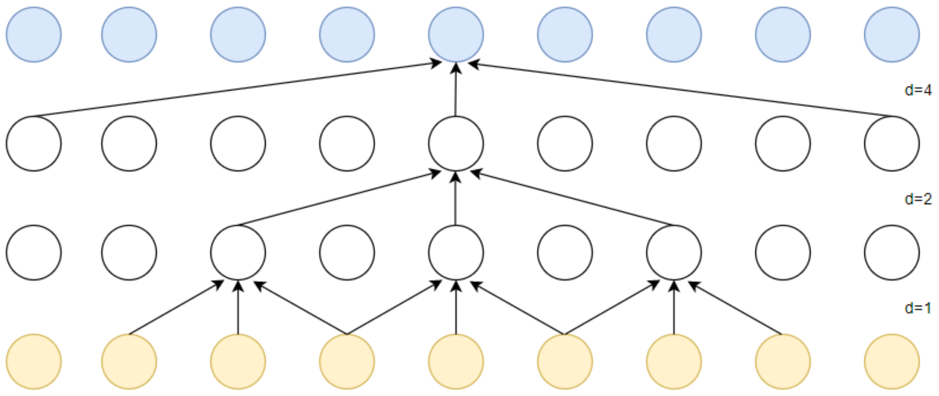
[6]Stride is the distance between spatial locations where the convolution kernel is applied (Gonzalez & Woods, 2018)



(a) Causal convolution



(b) Dilated causal convolution



(c) Dilated non-causal convolution

Figure 7: Illustration of dilated causal convolutions

7.6 Ensemble Learning

Ensemble learning is a performance-enhancing technique for learning algorithms, where a prediction model is built using a collection of simpler base models (Hastie, Tibshirani, & Friedman, 2009), each constructed to solve the same problem. (Huang, Xie, & Xiao, 2009) define an ensemble model in conjunction with machine learning as the following:

> *An ensemble in the context of machine learning can be broadly defined as a machine learning system that is constructed with a set of individual models working in parallel and whose outputs are combined with a decision fusion strategy to produce a single answer for a given problem.*

The motivating property of ensemble learning is the ability to combine a group of weak learners[7] into one strong learner, or expert learner, producing significantly improved results compared with usage of the individual learners alone. The learning algorithms that the ensemble comprises are often termed *member* learners. They can be any machine learning algorithm, such as neural networks, support vector machines or decision trees (Huang et al., 2009). The ensemble of learners can be generated using three different approaches; using member learners generated from several different classification or regression algorithms, termed *heterogeneous* ensembles, using member learners generated using the same algorithms presented with different training data, termed *homogeneous* ensembles, or a combination thereof.

The collective use of multiple learning algorithms to create a single model have shown to produce more stable and robust estimates than the results achieved using any of the individual learning algorithms alone (Kim, Xu, & Barber, 2020). Ensemble learners are able to produce very accurate results, since combining several models with relatively similar bias reduces the variance, thereby improving the accuracy (C. Zhang & Ma, 2012). Intuitively, the generalization ability of the combined model similarly strengthens. However, this is only the case if the member learners are sufficiently diverse and accurate. Diversity between member learners is essential for the ensemble performance, since little is gained by combining a vast ensemble of learners if they all produce the same result. The core idea of ensemble learning is to combine a set of diverse models that together produce satisfactory results by filling out each other's shortcomings, which is possible only if the models are adequately diverse and accurate.

The challenge in obtaining diversity in ensemble models lies in the fact that the member learners are all built for solving the same task, and usually trained on data derived from the same dataset, resulting in the member learners being highly correlated. When implementing the member learners, a trade-off between the individual learners' accuracy and the diversity between them exists. Combining very accurate but highly correlated learners often results in the ensemble learner performing worse than if some accurate and some weak performing learners are combined, since complementarity is more important than accuracy alone (Zhou, 2012).

[7]Weak learners are are learning algorithms than perform slightly better than a random guess.

The ensemble method is implemented as two subprocedures; first a population of base learners is trained using different data sets, or sampled *multisets* from the available training data. Next, the base learners are used to make predictions which then are combined to form a single predictor. A common ensemble learning architecture is shown in Fig. 8. The predictions from the base learners are combined using an aggregation function, where popular aggregation functions include the mean, median, or trimmed mean. Using different aggregation functions have different benefits, e.g. using the mean reduces the mean square error (MSE), sensitivity to outliers is prevented using the median, and the trimmed mean achieves a compromise of both (Xu & Xie, 2020). The homogeneous ensemble method is formalized in Algorithm 1.

Algorithm 1: Ensemble Learning - Homogeneous Ensembles

```
input  : Data $\{(x_i, y_i)\}_{i=1}^{n}$, base learning algorithm $A$, aggregation function $\phi$
output: Ensemble regression function $\hat{\mu}_{\phi}$
1 for $b = 1, \ldots, B$ do
2     Sample an index set $S_b = (i_{b,1}, \ldots, i_{b,m})$ from indices $(1, \ldots, n)$ with or
        without replacement
3     Compute $\hat{\mu}_b = A((X_{ib,1}, Y_{ib,1}), \ldots, (X_{ib,m}, Y_{ib,m}))$
4 Define $\hat{\mu}_{\phi} = \phi(\hat{\mu}_1, \ldots, \hat{\mu}_B)$
```

The multisets created from the training data can be sampled using several different methods, e.g. *bootstrapping* or *subsampling* (Kim et al., 2020). Bootstrapping creates multiple samples from the original data by randomly sampling *with* replacement (Hastie et al., 2009), where the sampled multi-sets often are of the same size as the original dataset. Contrary to bootstrapping, subsampling crates multisets from the original data by extracting subsets with size smaller than the original dataset, *without* replacement. One of the earliest and simplest of the ensemble methods is bootstrap aggregating, or *bagging* for short. Bagging uses bootstrapping together with the mean aggregation function to create ensemble models. Bagging obtains diversity between member learners by training the individual learners on different samples of the same dataset. Subsample aggregating, commonly abbreviated as *subagging*, is a variant of bagging, where subsampling is used instead of bootstrap sampling.

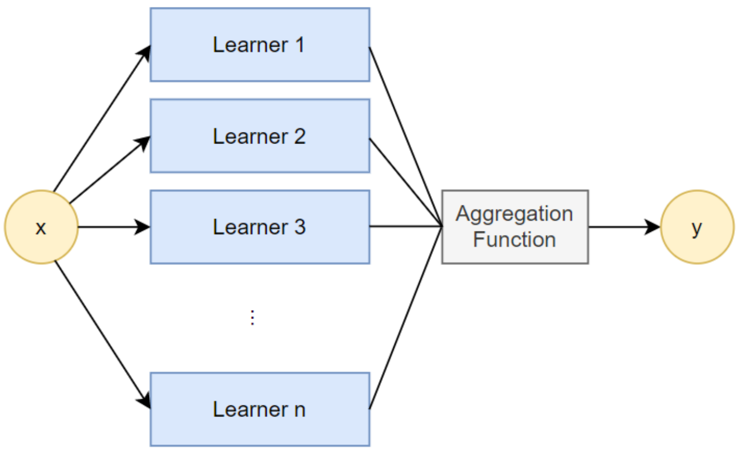


Figure 8: A ensemble learning architecture

8 Quantile Regression

Quantile regression, introduced by (Koenker & Bassett Jr, 1978), models the conditional distribution of a dependent variable, Y, given a selection of explanatory variables, X, in the form of quantiles (Nowotarski & Weron, 2015). A quantile divides a population into groups that correspond to a certain fraction of the population distribution (Gilchrist, 2000), e.g. the median of a sample corresponds to the 0.5 quantile. Instead of modeling a single value, like the conditional mean in the method of least squares, quantile regression aims to model a set of quantiles that can represent the entire conditional distribution. The conditional distribution function of Y given $X = x$ can be expressed as follows (Romano, Patterson, & Candès, 2019):

$$F(y|X = x) := \mathbb{P}\{Y \leq y|X = x\} \tag{19}$$

The α-th conditional quantile of the dependent variable, y, is the value

$$q_\alpha(x) = inf\{F(y|X = x) \geq \alpha\},$$

where the function $q_\alpha(x)$ is referred to as the conditional quantile function. An approximation of the entire probability distribution of y can be obtained from the quantile estimates in the range $(0 < \alpha < 1)$ (Taylor, 2000).

Non-parametric prediction intervals for Y_{n+1} can be obtained directly from the quantile function acquired from quantile regression (Gasthaus et al., 2019). In the same manner as ordinary regression aims to minimize the sum of squared error to predict a point estimate, quantile regression aims to minimize the quantile loss, also called the *pinball loss*, described below, to predict a given quantile in the range (0,1). Prediction intervals are constructed by estimating two conditional quantiles from the data, where the corresponding confidence level of the prediction interval is the difference between the two quantiles levels, i.e. the 90 % prediction interval can be obtained from the 0.05 and 0.95 quantile (Nowotarski & Weron, 2018), termed the lower and upper quantile, respectively. The estimated conditional prediction interval obtained from these two quantiles is defined as follows (Romano, Patterson, & Candès, 2019):

$$\hat{C}(x) = [\hat{q}_{\alpha_{lo}}(x), \hat{q}_{\alpha_{hi}}(x)] \tag{20}$$

where $\alpha_{lo} = \alpha/2 = 0.05$ and $\alpha_{hi} = 1 - \alpha/2 = 0.95$, and correspondingly $\hat{q}_{\alpha_{lo}}(x)$ and $\hat{q}_{\alpha_{hi}}(x)$ is the lower and upper estimated conditional quantile functions, forming the lower and upper bounds of the prediction interval. As seen in the equation, the width of the prediction interval is dependent on x, and can therefore significantly vary from sample to sample. Prediction intervals obtained using quantile regression are therefore adaptive to heteroscedastic data, e.g. time series with time-dependent variance (Shi, Worden, & Cross, 2019). Despite this useful property, the conditional interval above is a finite sample estimate of the ideal interval $C(x)$, and the actual coverage of the prediction interval is therefore not guaranteed to equal the designed confidence level α, i.e. $\mathbb{P}\{Y_{n+1} \in C(X_{n+1})\} \geq 1 - \alpha$.

8.1 Pinball Loss

The quantile functions used to construct prediction intervals must be estimated from the data. This estimation can be cast as a optimization problem, where a commonly used loss function is the *pinball loss*, a weighted absolute error measure, defined as (Wang, Gan, et al., 2019):

$$L_{q,t} = \begin{cases} (1-q)(\hat{y}_t^q - y_t), & \hat{y}_t^q \geq y_t \\ q(y_t - \hat{y}_t^q), & \hat{y}_t^q < y_t \end{cases} \tag{21}$$

where y_t denotes the sample label at time t, $\hat{y}_t^q$ is the estimated q-th quantile for the given target quantile level q. The quantile loss is asymmetric: if the predicted quantile is higher than the observed value, the penalty is multiplied by $(1-q)$, whereas if the predicted quantile is lower than the observed value, the penalty is multiplied by q. Therefore, for low-probability quantiles, the pinball loss function penalizes overestimation more than underestimation, and oppositely for high-probability quantiles (Hatalis, Lamadrid, Scheinberg, & Kishore, 2017).

The pinball loss expresses how well the estimated quantile relates to the actual distribution of the data, where a lower pinball loss indicates a more accurate estimation. To evaluate probabilistic forecasts derived from quantile functions, the pinball loss can be used as an evaluation metric (Hatalis et al., 2017). The full predictive densities can be evaluated by averaging the pinball loss score over all target quantiles for all observations in the forecasting horizon. The total average pinball loss is referred to as the quantile score, and is by (Grushka-Cockayne, Lichtendahl Jr, Jose, & Winkler, 2017) proven to be a *proper* scoring rule. Scoring rules provide a quality measure of probabilistic forecasts, obtained by appointing a numerical value based on the predictive distribution and the estimated value itself (Gneiting & Raftery, 2007). For probabilistic predictions in the form of intervals, proper scoring rules consider the prediction interval's coverage and length jointly (Gneiting, Balabdaoui, & Raftery, 2007).

For a single quantile level, the average pinball loss function is defined as:

$$L_q = \frac{1}{T}\sum_{t=1}^{T} L_{q,t}(y_t, \hat{y}_t^q) \tag{22}$$

where T is the number of observations, and $L_{q,t}$ is the pinball loss defined in Eq. (21). Changing the objective from minimizing a single quantile level to simultaneously minimizing Q different quantile levels, $\mathcal{Q} = \{q_1, q_2, \dots, q_Q\}$, results in the following equation for the quantile score:

$$L = \frac{1}{Q}\sum_{q=q_1}^{q_Q} L_q(y_t, \hat{y}_t^q) = \frac{1}{QT}\sum_{q=q_1}^{q_Q}\sum_{t=1}^{T} L_{q,t}(y_t, \hat{y}_t^q) \tag{23}$$

When the upper and lower quantile functions are simultaneously minimized using the equation above, narrow prediction intervals are rewarded, and a penalty dependent on the quantile level is incurred if the actual observation falls outside the interval.

The pinball loss was used as the performance measurement in the Global Energy Forecasting Competition 2014 (Hong et al., 2016), and it has become a popular performance measure for probabilistic forecasts obtained using quantiles. However, it is worth noticing that the pinball loss only conveys a *relative* performance measure, and data on different scales will correspondingly have different scaled pinball losses. Pinball loss scores can be placed on a similar scale by normalizing the data, which is convenient for comparison purposes.

9 Conformal Prediction

Conformal prediction, introduced by (Gammerman A., 1998), is an on-top probabilistic forecasting framework that can be used in conjunction with machine learning algorithms for both regression and classification problems. The probabilistic forecasts obtained from conformal predictors are given in the form of intervals, guaranteed to marginally satisfy the designed coverage level for finite samples (Romano, Patterson, & Candès, 2019).

Conformal prediction assumes that new observations are similar to previous examples, and constructs prediction intervals that with a given confidence contains the true value of a new observation. (Gammerman A., 1998) defines examples as successive pairs of observations and labels, $(X_1, Y_1), \ldots, (X_n, Y_n)$, $X_i \in \mathbb{R}^d, Y_i \in \mathbb{R}$, where the degree of similarity between the provided examples serves as a base for estimating the preciseness of the predictions of a new label Y_{n+1}, made by the underlying algorithm. The degree of similarity between examples is termed *conformity*, and can be defined using any distance measurement. In regression problems, the conformity score is often expressed by the absolute difference between the actual value and the predicted value. Conformal predictors make no hard assumptions about the distribution of the training examples, besides them being exchangeable. Assuming exchangeability means that the order in which the examples are presented to the model should not matter, and conformal predictors therefore assume that there are no temporal dependencies among the examples (Shafer & Vovk, 2008).

The goal of conformal prediction is to construct a prediction interval, i.e. a subset of $\mathbb{R}$ that contains Y_{n+1} with an associated degree of confidence, given X_{n+1} and the provided training examples. The interval must be of a size that ensures, with a given confidence, that the true label of the object falls within its range, with the requirement that as the significance level increases, the size of the interval must shrink. This requirement results in less specific predictions having a higher degree of confidence. A potential label with a small confidence signifies low conformity between the prediction and previous examples, whereas a prediction with a high confidence has high conformity, and should therefore be included in the prediction interval.

9.1 Inductive Conformal Prediction

The original theory on conformal prediction describes conformal prediction as a transductive inference method. Unlike inductive interference, where training data is used to produce a general model that can be employed to forecast values of a test dataset, transductive interference produces a forecast for the individual test sample directly by using the provided training data (Papadopoulos, 2008). To construct prediction intervals using transductive conformal prediction, each possible label must be considered, i.e. an infinite set of labels. In addition, the model needs to be retrained for each new test sample (Vovk, Gammerman, & Shafer, 2005). (Zeni, Fontana, & Vantini, 2020) report that methods for reducing the number of considered labels exist, making the computations more feasible, but still, the use of transductive conformal predictors remains computational expensive, making the conjunction with models that presents the samples to the model several times during training, e.g. neural networks, impracticable. A modification of conformal prediction, termed *inductive* or *split* conformal prediction[8], was presented by (Papadopoulos et al., 2002) to avoid the shortcomings of the original method, with the drawback that the training data must be split into two disjoint sets. As the name suggests, the new framework is based on inductive interference, which reduces the number of computations, making conformal prediction possible to combine with the aforementioned models.

For a regression problem, split conformal prediction first splits the training data into two subsets; the proper training set, $\mathcal{I}_1$, and a calibration set, $\mathcal{I}_2$. Secondly, the underlying regression model is fitted using the proper training set, and applied to the calibration set to construct predictions. The absolute difference, i.e. the absolute value of the residuals between the constructed predictions and the true labels in the calibration set are then computed, serving as the conformity measure. To build the prediction intervals for a given confidence level, α, the empirical quantile of the absolute residuals on the calibration set is calculated, and the prediction interval of a new data point X_{n+1} is given by (Romano, Patterson, & Candès, 2019):

$$C(X_{n+1}) = [\hat{\mu}(X_{n+1}) - Q_{1-\alpha}(R, \mathcal{I}_2), \hat{\mu}(X_{n+1}) + Q_{1-\alpha}(R, \mathcal{I}_2)] \tag{24}$$

where $\hat{\mu}(X_{n+1})$ is the prediction made by the underlying regression algorithm, R is the set of absolute residuals of the predictions constructed for the samples in the calibration set, where $R_i = |Y_i - \hat{\mu}(X_i)|$, and $Q_{1-\alpha}(R, \mathcal{I}_2)$ is defined as:

$$Q_{1-\alpha}(R, \mathcal{I}_2) = (1-\alpha)\text{-th empirical quantile of } R = \{R_i : i \in \mathcal{I}_2\} \tag{25}$$

(Vovk et al., 2005) proves that the prediction interval in Equation (24) is guaranteed to satisfy the marginal coverage $\mathbb{P}\{Y_{n+1} \in C(X_{n+1})\} \geq 1-\alpha$ for the given confidence level α.

The most appealing property of split conformal prediction is unarguably that the prediction intervals constructed are guaranteed to marginally satisfy the designed coverage rate for finite samples, unlike the intervals obtained from quantile regression, described in Section 8. However, drawbacks of this exists; as shown in Equation (24), the prediction intervals are symmetric, and their length at a new test point, X_{n+1}, is fixed and equal to $2Q_{1-\alpha}(R, \mathcal{I}_2)$, hence independent of X_{n+1}. Intervals of fixed length are often unnecessarily conservative and do not adapt to local variability (Romano, Patterson, & Candès, 2019). *Normalized* conformal prediction, an extension of inductive conformal prediction, addresses the problem of fixed-length prediction intervals by replacing the absolute residuals with the scaled residuals:

$$\tilde{R}_i = \frac{|Y_i - \hat{\mu}(X_i)|}{\hat{\sigma}(X_i)} = \frac{R_i}{\hat{\sigma}(X_i)} \quad i \in \mathcal{I}_2,$$

[8]Since this thesis aims to construct probabilistic forecasts for time series using neural networks, only inductive conformal prediction is considered. Therefore, throughout the rest of this thesis, split conformal prediction is simply referred to as conformal prediction.

where $\hat{\sigma}(X_i)$ is a measure of the dispersion of the residuals at X_i, fitted on the proper training set. In normalized split conformal prediction, the prediction interval in Eq. (24) modifies to (Romano, Patterson, & Candès, 2019):

$$C(X_{n+1}) = [\hat{\mu}(X_{n+1}) - \hat{\sigma}(X_i)Q_{1-\alpha}(\tilde{R}, \mathcal{I}_2), \hat{\mu}(X_{n+1}) + \hat{\sigma}(X_i)Q_{1-\alpha}(\tilde{R}, \mathcal{I}_2)] \tag{26}$$

Although the normalized conformal prediction framework solves the problem of fixed length prediction intervals, limitations exist; extra variability is introduced by estimating $\hat{\sigma}$ in addition to $\hat{\mu}$, which can lead to inflated prediction intervals if the data is in fact homoscedastic. Furthermore, $\hat{\sigma}$ is fitted using the absolute residuals on the proper training set, and there exists a crucial difference between the absolute residuals of the proper training set and the calibration set; the former are biased by an optimization procedure designed to minimize them, while the latter are unbiased (Romano, Patterson, & Candès, 2019). For models such as neural networks, designed to minimize the residuals of the training data, the training residuals are often poor estimates of the test residuals, resulting in the normalized conformal prediction framework being less adaptive than designed to be.

Indeed, the introduction of inductive inference allows conformal prediction to be used in combination with methods that require the data to be presented to the model several times, significantly broadening the area of use. Despite this, both transductive and inductive-based conformal prediction assumes exchangeability, making conformal predictors not well suited for the problem of time series forecasting (Kath & Ziel, 2020), where the order of the observations clearly matters. Yet, conformal prediction has been applied in the field of time series forecasting, where the assumption of exchangeability is simulated by splitting the data into the three disjoint datasets (train, calibration and test) randomly, resulting in the three datasets having approximately identically distributed conformity scores, as they would if they were truly exchangeable. This approach allows conformal prediction to be used for time series forecasting problems. However, by removing the additional knowledge one obtains via the temporal ordering of the samples, important information is lost and time series forecasting algorithms that rely on this information, e.g. RNNs and TCNs, can therefore not be utilized.

Part III / Proposed Method

The aim of this thesis is to answer the research questions formulated in Section 4:

- *Is it possible to combine conformal prediction and quantile regression to construct a probabilistic forecasting method that inherits the advantages of both techniques?*
- *Can conformal predictors be applied to the problem of time series forecasting while preserving the temporal ordering of the observations?*

This part of the thesis presents the proposed method, after reviewing the two related works that served as a foundation for the methodology developed. In particular, the proposed method is heavily inspired by the algorithms presented in the articles *Conformalized Quantile Regression* by (Romano, Patterson, & Candès, 2019), and *Conformal prediction interval for dynamic time-series* by (Xu & Xie, 2020), who tackle above-stated problems separately. Sections 10 and 11 introduce these two algorithms, respectively, discussing both their validity and the assumptions made. Section 12 presents the proposed model, which combines the theory presented in the two papers to construct a probabilistic electricity load forecasting method that produces marginally valid prediction intervals adaptive to local variability within time series data.

10 Conformalized Quantile Regression

(Romano, Patterson, & Candès, 2019) present an algorithm termed *Conformalized quantile regression* (CQR)[9], a probabilistic forecasting method that combines conformal prediction and quantile regression to construct prediction intervals fully adaptive to heteroscedasticity while guaranteeing valid marginal coverage. The authors report that the motive behind the combination of conformal predictors and quantile regression is that existing conformal methods can be unnecessarily conservative, producing intervals with constant or weakly varying lengths. The use of quantile regression together with conformal prediction inherits the advantages of both; the properties of quantile regression allows the method to adapt to local variability, and the use of conformal prediction guarantees that the actual coverage matches the designed coverage level, $\mathbb{P}\{Y_{n+1} \in C(X_{n+1})\} \geq 1 - \alpha$.

Similarly to split conformal prediction, the CQR algorithm, summarized in Algorithm 2, assumes the samples to be exchangeable, and starts by splitting the data into a proper training set, $\mathcal{I}_1$, and a calibration set, $\mathcal{I}_2$. Next, a quantile regression algorithm is fitted to the proper training set, constructing two conditional quantile functions, $\hat{q}_{\alpha_{lo}}$ and $\hat{q}_{\alpha_h i}$, defining the lower and upper limits of the prediction intervals, respectively. The conditional quantile functions are used to construct intervals as in Eq. (20). Lastly, the obtained prediction interval is *conformalized* using an approach similar as for split conformal prediction in Eq. (24), substituting the point prediction with the upper and lower quantile functions, and the absolute error conformity score with the following:

$$E_i = \max\{\hat{q}_{\alpha_{lo}}(X_i) - Y_i, Y_i - \hat{q}_{\alpha_{hi}}(X_i)\} \tag{27}$$

[9]https://github.com/yromano/cqr

This conformity score quantifies the error made by the prediction intervals obtained using quantile regression. If the actual observation falls below the lower prediction interval bound, i.e. $Y_i < \hat{q}_{\alpha_{lo}}$, then the conformity score is the magnitude of the error, $E_i = |Y_i - \hat{q}_{\alpha_{lo}}(X_i)|$. Contrarily, if the actual observation lies above the upper bound of the prediction interval, $E_i = |Y_i - \hat{q}_{\alpha_{hi}}(X_i)|$. Lastly, if the actual observation is contained within the interval $[\hat{q}_{\alpha_{lo}}(X_i), \hat{q}_{\alpha_{hi}}(X_i)]$, E_i is the larger of the two negative numbers $\hat{q}_{\alpha_{lo}}(X_i) - Y_i$ and $Y_i - \hat{q}_{\alpha_{hi}}(X_i)$. By defining the conformity score in this way, both under- and overcoverage is accounted for, and the quantile regression prediction interval length can be both extended and reduced. The authors also present an extension of the conformalization step, introducing an asymmetrical conformity score that controls the coverage errors of $\hat{q}_{\alpha_{lo}}$ and $\hat{q}_{\alpha_{hi}}$ independently. The asymmetric conformity score results in a stronger coverage guarantee, but can increase the width of the intervals.

Algorithm 2: Conformalized Quantile Regression

```
input : Training data $(X_i, Y_i) \in \mathbb{R}^p \times \mathbb{R}, 1 \leq i \leq n$
        Confidence level $\alpha \in (0, 1)$
        Quantile regression algorithm $\mathcal{A}$
output: prediction intervals $C(x)$
1 Randomly split $\{1, \ldots, n\}$ into two disjoint sets $\mathcal{I}_1$ and $\mathcal{I}_2$.
2 Fit two conditional quantile functions: $\{\hat{q}_{lo}, \hat{q}_{hi}\} \leftarrow \mathcal{A}(\{X_i, Y_i) : i \in \mathcal{I}_1\})$.
3 Compute $E_i$ for each $i \in \mathcal{I}_2$, as in Eq. 27.
4 Compute $Q_{1-\alpha}(E, \mathcal{I}_2)$, the $(1-\alpha)(1 + 1/|\mathcal{I}_2|)$-th empirical quantile of
    $\{E_i : i \in \mathcal{I}_2\}$.
5 Return $\hat{C}(x) = [\hat{q}_{\alpha_{lo}}(x) - Q_{1-\alpha}(E, \mathcal{I}_2), \hat{q}_{\alpha_{hi}}(x) + Q_{1-\alpha}(E, \mathcal{I}_2)]$ for unseen input
    $X_{n+1} = x$.
```

Note that the value of $Q_{1-\alpha}(E, \mathcal{I}_2)$ used to conformalize the prediction intervals constructed by the quantile regression algorithm is fixed for all future points X_{n+1}, similarly to $Q_{1-\alpha}(R, \mathcal{I}_2)$ in split conformal prediction.

10.1 Theoretical Analysis

This subsection briefly discusses the assumptions and validity of the CQR method. For a more thorough and in-depth discussion, the reader is referred to the paper by (Romano, Patterson, & Candès, 2019).

The CQR algorithm assumes the sample pairs $(X_i, Y_i), i = 1, \ldots, n+1$ to be exchangeable, just as conformal predictors, and constructs prediction intervals with the same marginal coverage guarantee, regardless of the distribution of the data. Additionally, the authors report that if the conformity scores calculated using Eq. (27) are almost surely distinct[10], the resulting intervals are nearly perfectly calibrated, meaning that the actual coverage of the prediction interval is almost identical to its designed coverage level. Having a coverage close to the designed coverage level assures that valid coverage is obtained, and can avoid overly wide prediction intervals by reducing the occurrence of *overcoverage*. For

[10]Distinct conformity scores are scores whose values only occur once for each $i \in \mathcal{I}_2$.

overly wide prediction intervals, the actual coverage level can be significantly above the designed coverage level, which may not always be preferable. Generally, when constructing prediction intervals, the actual coverage should be approximately equal to the designed coverage, avoiding both undercoverage and overcoverage to assure that the prediction intervals are representative and informative.

In the experiments, the authors focus on CQR in combination with quantile regression neural networks and quantile regression forests, and remarks that their experiments have shown that when using quantile neural networks as the underlying regression algorithm, the intervals tend to be too conservative, constructing unnecessarily wide prediction intervals. They avoid this problem by tuning the nominal quantile levels of underlying quantile neural networks as additional hyperparameters, which is proven not to invalidate the coverage guarantee.

11 Ensemble Batch Prediction Intervals

(Xu & Xie, 2020) present a conformal prediction-inspired method for building distribution-free prediction intervals for time series, termed *Ensemble Batch Prediction Intervals*, or *EnbPI* for short, reporting that their method is suitable for non-stationary, dynamic time series. As described in Section 9, conformal predictors based on either transductive or inductive interference assume the samples to be exchangeable, making them unsuitable for time series. Contrarily, the EnbPI method does not assume exchangeability; instead, it places mild assumptions on the error process and the accuracy of the underlying regression algorithm, and can therefore be applied to time series data. Additionally, the method does not require data splitting as in split conformal prediction, which is advantageous for small-sample problems.

The EnbPI method, summarized in Algorithm 3, constructs probabilistic forecasts by aggregating point forecasts produced using bootstrap ensemble estimators. The ensemble estimators produce predictions by applying a regression algorithm to bootstrapped samples drawn from the training data and aggregating the results into a single prediction using the mean aggregation function. The method assumes that the samples, (x_t, y_t), are generated according to a model on the form

$$Y_t = f(X_t) + \epsilon_t, \quad t = 1, 2, 3, \ldots \tag{28}$$

The goal of the underlying regression algorithm is to estimate the function f using a *leave-one-out* estimator $\hat{f}_{-t}$, where the $\hat{f}_{-i}$ leave-one-out estimator excludes the i-th training sample (x_i, y_i) from its training dataset. The prediction intervals produced by the EnbPI algorithm are on the following form:

$$C^{\alpha}_{T,t} = \hat{f}_{-t}(x_t) \pm (1 - \alpha) \text{ quantile of } \{\hat{\epsilon}_i\}_{i=t-1}^{t-T} \tag{29}$$

The symmetric prediction interval is centered at the point prediction $\hat{f}_{-t}(x_t)$, with a width equal to the $(1 - \alpha)$-th empirical quantile of the latest T available residuals, i.e. the quantile of a list with length T, indexed by i. The residuals are calculated using the absolute error between the training sample labels and the leave-one-out estimators as the conformity score, defined as

$$\hat{\epsilon}_i = |y_i - \hat{f}_{-i}(x_i)|.$$

Algorithm 3: Ensemble Batch Prediction Intervals (EnbPI)

```
input  : Training data ${(x_i, y_i)}_{i=1}^{T}$, regression algorithm $\mathcal{A}$, confidence level $\alpha$,
         aggregation function $\phi$, number of bootstrap models $B$, batch size $s$, and test
         data $\{(x_i, y_i)\}_{t=T+1}^{T+T_1}$, with $y_t$ revealed only after the batch of $s$ prediction
         intervals with $t$ in the batch are constructed.
output: Ensemble prediction intervals $\{C_{T,t}^{\phi,\alpha}(x_t)\}_{t=T+1}^{T+T_1}$
1  for $b = 1, \ldots, B$ do
2      Sample with replacement an index set $S_b = (i_1, \ldots, i_T)$ from indices $(1, \ldots, T)$
3      Compute bootstrap estimator $\hat{f}^b = \mathcal{A}(\{(x_i, y_i) | i \in S_b\})$
4  Initialize $\boldsymbol{\epsilon} = \{\}$
5  for $i = 1, \ldots, T$ do
6      $\hat{f}_{-i}^{\phi}(x_i) = \phi(\{\hat{f}^b(x_i) | i \notin S_b\})$
7      Compute $\hat{\epsilon}_i^{\phi} = |y_i - \hat{f}_{-i}^{\phi}(x_i)|$
8      $\boldsymbol{\epsilon} = \boldsymbol{\epsilon} \cup \{\hat{\epsilon}_i^{\phi}\}$
9  for $t = T + 1, \ldots, T + T_1$ do
10     Let $\hat{f}_{-t}(x_t) = \phi(\{\hat{f}^b(x_t)\}_{b=1}^{B})$
11     Let $\omega_t^{\phi} = (1 - \alpha)$ quantile of $\boldsymbol{\epsilon}$
12     Return $C_{T,t}^{\phi,\alpha}(x_t) = [\hat{f}_{-t}(x_t) \pm \omega_t^{\phi}]$
13     if $t - T = 0$ $mod$ $s$ then
14         for $j = t - s, \ldots, t - 1$ do
15             Compute $\hat{\epsilon}_j^{\phi} = |y_j - \hat{f}_{-j}^{\phi}(x_t)|$
16             $\boldsymbol{\epsilon} = (\boldsymbol{\epsilon} - \{\hat{\epsilon}_{:s}^{\phi}\}) \cup \{\hat{\epsilon}_j^{\phi}\}$ and reset index of $\boldsymbol{\epsilon}$
```

Similarly to conformal prediction, the EnbPI method is used in an online setting, but includes a batch size parameter, s, determining the rate at which the model receives feedback. The feedback allows the method to be adaptive to dynamic time series, while only being fitted once, done by updating the list of available residuals after each s predicted time steps. When the model receives feedback, the list containing the latest T available residuals is updated, where the s new absolute residuals between the predicted and actual observations in the test dataset are added, and the s earliest residuals are removed. For $s = 1$, the prediction intervals are built sequentially, presenting the model with a new sample point (x, y) immediately, but can be increased, i.e. $s > 1$. If the model never receives feedback, i.e. $s = \infty$, the prediction intervals are all based on the training residuals, resulting in the intervals for all time steps in the test data having equal width. Producing prediction intervals with a fixed length is often unsatisfactory, and the authors report that the batch size parameter should therefore be kept as small as possible, but its value should be dependent on the data collection process.

11.1 Theoretical Analysis

This subsection briefly discusses the assumptions and validity of the EnbPI method. For a more thorough and in-depth discussion, the reader is referred to Section 4 in the paper by (Xu & Xie, 2020).

As mentioned above, the time series data generating process by the EnbPI algorithm is assumed to follow a model on the form:

$$Y_t = f(X_t) + \epsilon_t, \quad t = 1, 2, 3, \ldots,$$

where mild assumptions on the time series' stochastic errors and the underlying regression algorithms are made. The error process $\{\epsilon_t\}_{t\geq 1}$ is assumed to be stationary and strongly mixing, replacing the exchangeability assumption required by conformal predictors. The term strong mixing was introduced by (M. Rosenblatt, 1956), and refers to asymptotic independence. A stochastic process is strongly mixing if the dependence between $X(t)$ and $X(t+T)$ goes to zero as the number of time steps between the two observations increases. The authors state that a highly non-stationary time series that exhibit arbitrary dependence still can be strongly mixing, or even have independent and identically distributed errors, and argue that the assumption made on the time series' error process is mild and general, even verifiable (Xu & Xie, 2020).

Further, the estimated errors, $\hat{\epsilon_t}$, are assumed to be close to the true errors, ϵ_t. For this assumption to be valid, overfitting must be avoided. To assure that the estimated residuals resemble the test residuals, out-of-sample training residuals obtained via leave-one-out training estimators are used. Some regression algorithms, such as neural networks, construct the optimal model by finding the model parameters that minimize the training error. The in-sample training errors are often small compared to out-of-sample errors, and by using the out-of-sample training residual during the construction of the prediction intervals in Eq. (29), unrepresentative residuals are avoided.

The ensemble learners are used to estimate the unknown model f. The ensemble regression algorithms are only trained once and are used to predict the center of the prediction intervals for the future time steps. Hence, the assumption placed on the ensemble learners is that they must model f with satisfactory accuracy. The authors report that in reality, this assumption can fail when the batch size parameter, s, is large and time steps far into the future are predicted. The characteristics of non-stationary, dynamic time series can significantly change over time, reaching change points that alter the underlying model f, resulting in the predictions of the ensemble models being unrepresentative for the new f for $t > T$. However, valid coverage can still be obtained if a small batch size parameter is used, but the resulting intervals become inflated if the out-of-sample absolute residuals are large.

The EnbPI algorithm constructs approximately marginally valid prediction intervals, if the assumptions made about the error process and the underlying regression algorithms hold, and does so without assuming data exchangeability, which makes it suitable for time series data.

12 Proposed Method: Ensemble Conformalized Quantile Regression

12.1 Motivation

Construction of probabilistic forecasts using quantile regression has the advantage of creating prediction intervals dependent on the input sample, thus allowing the length of the intervals to vary for each sample point, a significant advantage when predicting heteroscedastic data. Additionally, quantile regression does not assume a particular parametric distribution for the dependent variable, which makes the framework more robust and applicable to real-life situations, where one often lacks sufficient knowledge about the distribution of interest (Wen, Torkkola, Narayanaswamy, & Madeka, 2017). Quantile regression can easily be performed with neural networks guided by the pinball loss (Romano, Patterson, & Candès, 2019; Wang, Gan, et al., 2019; Elvers, Voß, & Albayrak, 2019), but the resulting predictions are not guaranteed to meet the specified coverage level, as the prediction intervals are based on quantile functions estimated from the data. Further, (Keren et al., 2018) reports that neural networks tend to produce overly confident non-calibrated prediction intervals; i.e. very narrow intervals, signifying high confidence, but where the actual observation fall within its boundaries on average only α_0 of the times, where $\alpha_0 < \alpha$, α being the designed confidence level. Similarly, (Romano, Patterson, & Candès, 2019) states that the coverage of quantile-based neural networks depends greatly on the hyperparameter tuning, where the actual coverage can range from 50% to 95% for different network configurations, for a specified coverage level of 90%. This lack of robustness motivates the need for using probabilistic frameworks producing calibrated prediction intervals, such as conformal predictors.

Conformal predictors produce distribution-free symmetric intervals that, on average, are guaranteed to cover the actual observations with the designed confidence. Despite this appeal, conformal predictors tend to be unnecessarily conservative, constructing prediction intervals with constant or weakly varying length (Romano, Patterson, & Candès, 2019). From Eq. (24), which defines the conformal prediction interval, it is clear that conformal predictors were designed with homoscedastic data in mind, as the intervals are constructed by estimating the expected value and building a fixed-width band around it (Sesia & Candès, 2020). In order to guarantee valid coverage when the data is in fact heteroscedastic, the length of the intervals must significantly increase to include the points furthest from the expected value. For symmetric intervals, the length increases identically in both directions, which can be undesirable since heteroscedastic data do not necessarily vary equally in both directions. Figure 15c and 23c show the hourly variability in electricity load for two different time series, clearly showing heteroscedasticity where the variability is greater in the upwards direction during the central hours of the day. This indicates that forecasting methods adaptive to such behavior should be applied for electricity load prediction intervals to be as short as possible. The advantages and disadvantages of conformal prediction and quantile regression are summarized in Table 1.

From the previous discussion, the need for a model that combines the properties of quantile neural networks and conformal predictors is evident, being adaptable to local variability while guaranteeing valid marginal coverage. The value added if such a method can also be applied to time series prediction with deep neural networks is considerable, given the prominent positions of these models in the recent research literature. The proposed method combines the theory behind the EnbPI and CQR algorithms, using an ensemble of quantile regression neural networks to construct prediction intervals that are conformalized using a method inspired by conformal predictors.

Table 1: A comparison of the advantages and disadvantages of conformal prediction and quantile regression.

	Conformal prediction	**Quantile regression**
Advantages	Marginal coverage guarantee	Adaptive to heteroscedasticity
Disadvantages	Constructs PIs with fixed or weakly varying lengths	No coverage guarantee for finite samples

12.2 Model Overview

As introduced above, the proposed method, termed *Ensemble Conformalized Quantile regression* (EnCQR), combines an ensemble of pinball loss guided neural networks producing prediction intervals using quantile regression and conformal predictors. The overall network structure is illustrated in Fig. 9. The EnCQR method utilizes the out-of-sample ensemble prediction theory introduced in the EnbPI algorithm to apply conformal prediction to time series data, but replaces the aggregated point prediction with aggregated quantile predictions that produce the upper and lower bound of the prediction intervals. The resulting prediction intervals are conformalized using conformal predictors, as in the CQR algorithm.

In a paper closely related to the *Conformalized Quantile Regression* article by (Romano, Patterson, & Candès, 2019), (Romano, Barber, Sabatti, & Candès, 2019) further explore the asymmetric conformity score, briefly explained in the CQR article. The asymmetric conformity score is defined as follows:

$$\begin{aligned} E_{lo} &= \hat{q}_{\alpha_{lo}} - y \\ E_{hi} &= y - \hat{q}_{\alpha_{hi}} \end{aligned} \tag{30}$$

The motivation for defining the asymmetric conformity score is that the distribution of the conformity scores for the two estimated conditional quantile functions can be skewed, resulting in the prediction interval coverage error being asymmetrically spread over the left and right tails. If this occurs, the intervals can be wrongly confomalized, resulting in coverage below the target level. The asymmetric conformity score solves this problem by controlling the coverage of the two quantile functions independently, resulting in the following modification of the equation for the conformalized prediction interval:

$$\hat{C}(X_{n+1}) = [\hat{q}_{\alpha_{lo}}(X_{n+1}) - Q_{1-\alpha_{lo}}(E_{lo}, \mathcal{I}_2), \hat{q}_{\alpha_{hi}}(X_{n+1}) + Q_{1-\alpha_{hi}}(E_{hi}, \mathcal{I}_2)], \tag{31}$$

where $Q_{1-\alpha_{lo}}(E_{lo}, \mathcal{I}_2)$ is the $(1-\alpha_{lo})$-th emprical quantile of $\{E_{lo_i} : i \in \mathcal{I}_2\}$, and equivalently for $Q_{1-\alpha_{hi}}(E_{hi}, \mathcal{I}_2)$. The authors prove that the interval in Eq. (31) obtains a stronger coverage guarantee than the original interval, at the possible expense of slightly wider intervals. The proposed EnCQR algorithm is implemented using the asymmetrical conformity score, contrarily to the original CQR algorithm.

The batch-size parameter in the EnbPI algorithm enables the method to be used for dynamic time series data, leveraging new data without refitting the underlying regression algorithms. The list containing the out-of-sample residuals is updated after every s prediction, allowing the width of the subsequent prediction intervals to vary, and making the calibration of the prediction interval widths more dynamic and accurate (Xu & Xie, 2020). The update of the residuals empowers the algorithm to produce intervals that cover the true observation even if the aggregated prediction $\hat{f}_{-t}(x_t)$ is inaccurate, as described in Section 11.1. By utilizing the same approach as in the EnbPI algorithm, the conformity scores used to conformalize the quantile regression intervals in the EnCQR algorithm are updated after every s prediction, possibly allowing the method to adapt even more to local variability, compared to CQR (where the $(1-\alpha)$-th empirical quantile of $\boldsymbol{E}$ is fixed for all future points X_{n+1}). Additionally, the use of ensemble estimators avoids data splitting and the assumption of exchangeability, making the method suitable for time series data.

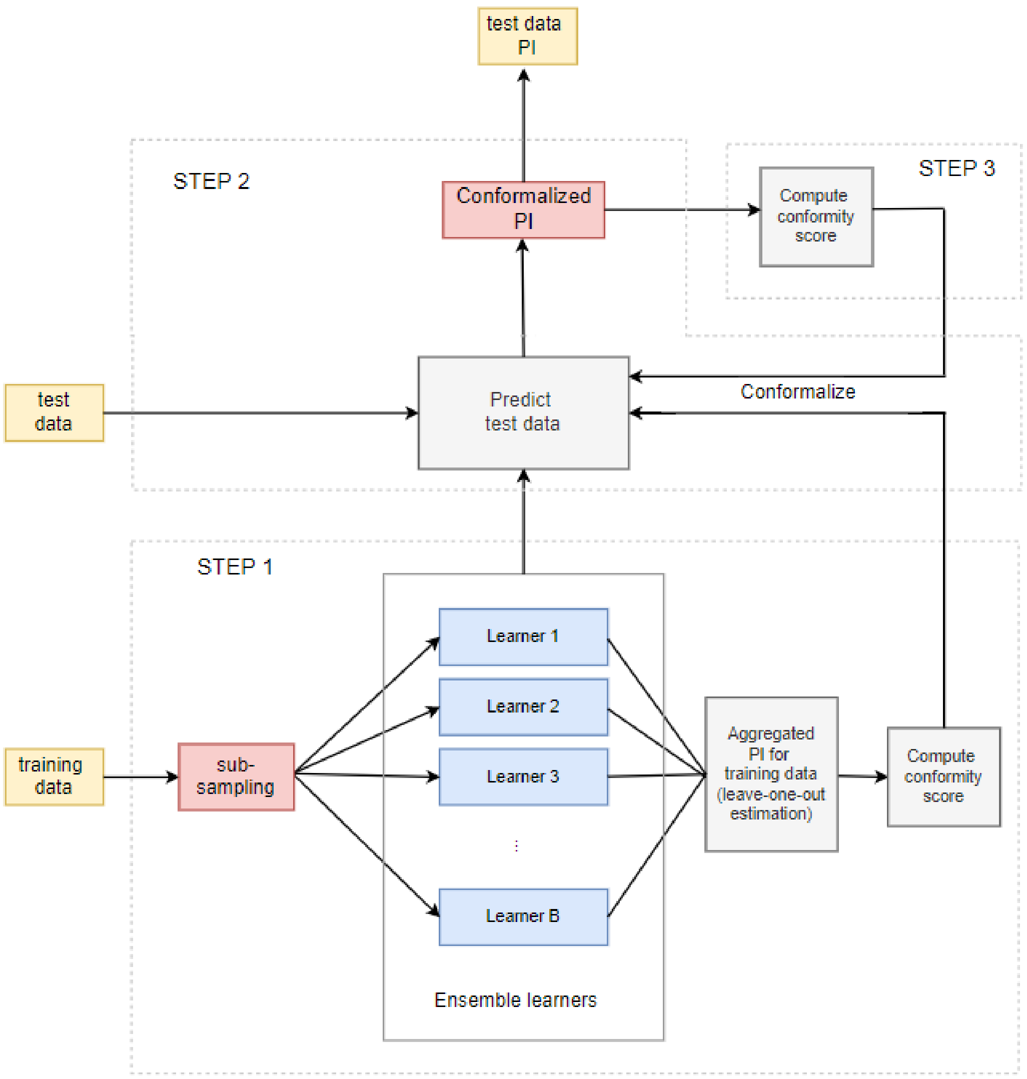


Figure 9: Ensemble conformalized quantile regression model overview. The algorithm uses an ensemble of underlying regression networks producing prediction intervals, conformalized using conformal prediction.

As Fig. 9 shows, the proposed method consists of three steps:

- **Step 1:** The first step is to train the ensemble member learners. Homogeneous learners are constructed using subsampling of the original training dataset: The training dataset consists of T observations and is divided into B equal subsets with size T/B. The subsets are referred to as S_b for $b = 1, \ldots, B$. As in the EnbPI algorithm, each ensemble model, $\hat{f}^b$, is trained on one of the ensemble subsets, S_b, resulting in each time step in the training data having $B-1$ models not trained on data containing that time step. For each sample x_i in $i = 1, \ldots, T$, $B-1$ prediction intervals can therefore be constructed using the ensemble estimators, and aggregated to form a single prediction interval for each y_i. Next, the asymmetric conformity scores defined in Eq. (30) are computed for all observations in the training data and appended to two lists $\boldsymbol{E}_{lo}$ and $\boldsymbol{E}_{hi}$.

- **Step 2:** After the ensemble learners have been trained, each observation in the test data is predicted using the ensemble learners, constructing a set of B estimated quantile functions for both the upper and lower prediction interval limit. The final prediction interval limits are found by aggregating together the estimated quantile functions for each quantile level using the mean aggregation function, which are conformalized using the $(1-\alpha_{lo})$ and $(1-\alpha_{hi})$-th quantile of the out-of-sample residuals, calculated during training, for the upper and lower quantile functions, respectively.

- **Step 3:** The final step in the algorithm is to update the lists containing the out-of-sample residuals after every s observation, done in the manner explained in Section 11, but now using the asymmetric conformity score. The batch size parameter can be a number between 1 and the total observation in the test data. If $s = 1$, the out-of-sample residuals are updated after each prediction, removing the first elements of the lists such that the length of $\boldsymbol{E}_{lo}$ and $\boldsymbol{E}_{hi}$ is always equal to T.

The EnCQR procedure is summarized in Algorithm 4.

Algorithm 4: Ensemble Conformalized Quantile Regression (EnCQR)

```
input  : Training data $\{(x_i, y_i)\}_{i=1}^{T}$, quantile regression algorithm $A$, confidence level
         $\alpha \in (0,1)$, aggregation function $\phi$, number of bootstrap models $B$, batch size
         $s$, and test data $\{(x_i, y_i)\}_{t=T+1}^{T+T_1}$, with $y_t$ revealed only after the batch of $s$
         prediction intervals with $t$ in the batch are constructed.
output: Ensemble prediction intervals $\{C_{T,t}^{\phi,\alpha}(x_t)\}_{t=T+1}^{T+T_1}$
1  for $b = 1, \dots, B$ do
2      Compute length of ensemble subset $t_b = T/B$
3      Sample an index set $S_b = (i_{t_b \times b}, \dots, i_{t_b \times b + t_b})$ from indices $(1, \dots, T)$
4      Fit quantile bootstrap estimator $\{\hat{q}_{\alpha_{lo}}^{b}, \hat{q}_{\alpha_{hi}}^{b}\} = A(\{(x_i, y_i) | i \in S_b\})$
5  Initialize $\boldsymbol{E}_{lo} = \{\}$
6  Initialize $\boldsymbol{E}_{hi} = \{\}$
7  for $i = 1, \dots, T$ do
8      $\{\hat{q}_{\alpha_{lo-i}}^{\phi}(x_i), \hat{q}_{\alpha_{hi-i}}^{\phi}(x_i)\} = \{\phi(\{\hat{q}_{\alpha_{lo}}^{b}(x_i)\}), \phi(\{\hat{q}_{\alpha_{hi}}^{b}(x_i)\})\}$ for $i \notin S_b$
9      Compute $E_{lo_i}$ and $E_{hi_i}$ as in Equation (30)
10     $\boldsymbol{E}_{lo} = \boldsymbol{E}_{lo} \cup \{E_{lo_i}\}$
11     $\boldsymbol{E}_{hi} = \boldsymbol{E}_{hi} \cup \{E_{hi_i}\}$
12
13 for $t = T+1, \dots, T+T_1$ do
14     Let $\hat{q}_{\alpha_{lo-t}}(x_t) = \phi(\{\hat{q}_{\alpha_{lo}}^{b}(x_t)\}_{b=1}^{B})$
15     Let $\hat{q}_{\alpha_{hi-t}}(x_t) = \phi(\{\hat{q}_{\alpha_{hi}}^{b}(x_t)\}_{b=1}^{B})$
16     Let $\omega_{lo_t}^{\phi} = (1 - \alpha_{lo})$ quantile of $\boldsymbol{E}_{lo}$
17     Let $\omega_{hi_t}^{\phi} = (1 - \alpha_{hi})$ quantile of $\boldsymbol{E}_{hi}$
18     Return $C_{T,t}^{\phi,\alpha}(x_t) = [\hat{q}_{\alpha_{lo-t}}(x_t) - \omega_{lo_t}^{\phi}, \hat{q}_{\alpha_{hi-t}}(x_t) + \omega_{hi_t}^{\phi}]$
19     if $t - T = 0 \ mod \ s$ then
20         for $j = t - s, \dots, t - 1$ do
21             Compute $E_{lo_j}$ and $E_{hi_j}$ as in Eq. (30)
22             $\boldsymbol{E}_{lo} = (\boldsymbol{E}_{lo} - \{E_{lo:s}\}) \cup \{E_{lo_j}\}$ and reset index of $\boldsymbol{E}_{lo}$
23             $\boldsymbol{E}_{hi} = (\boldsymbol{E}_{hi} - \{E_{hi:s}\}) \cup \{E_{hi_j}\}$ and reset index of $\boldsymbol{E}_{hi}$
```

12.2.1 Regression Network Architecture

The proposed forecasting method can be used together with any neural network type performing quantile regression. For time series forecasting, neural network types specifically designed to process sequential data are mostly used. Therefore, the use of two different underlying regression algorithms are considered: TCN and LSTM. The architectural details of the two network types used in the experiments are presented below.

- **TCN**: The TCN setup, inspired by the DeepTCN[11] network presented by (Chen et al., 2020), consist of several stacked residual blocks containing dilated convolutional layers, followed by a final fully-connected layer to map the output from the residual blocks into quantile prediction. The residual blocks, illustrated in Fig. 10, consist of two identical dilated causal convolutional layers, both followed by a batch normalization layer and ReLU activation. Depending on the number of residual blocks added in the network, the residual block's output is either passed as input to another residual block or to the final fully-connected output layer.

  Convolution-related hyperparameters, such as kernel size, number of residual blocks, and dilation rate are fixed and selected according to the size of the input samples in the different datasets. The most important principle for choosing kernel size and dilation length is to make sure that the network has a sufficiently large receptive field to capture most of the temporal dependencies in the input data.

  Contrarily to the skip connections in residual networks, the skip connections in the TCN residual blocks contain a 1×1 convolutional layer with number of filters equal to the convolutional layers within the residual block, prior to the element-wise addition operation $\oplus$. The additional convolutional layer is included to ensure that the addition operator receives tensors of the same shape, as the input and output of the TCN residual block can have different widths (Bai et al., 2018).

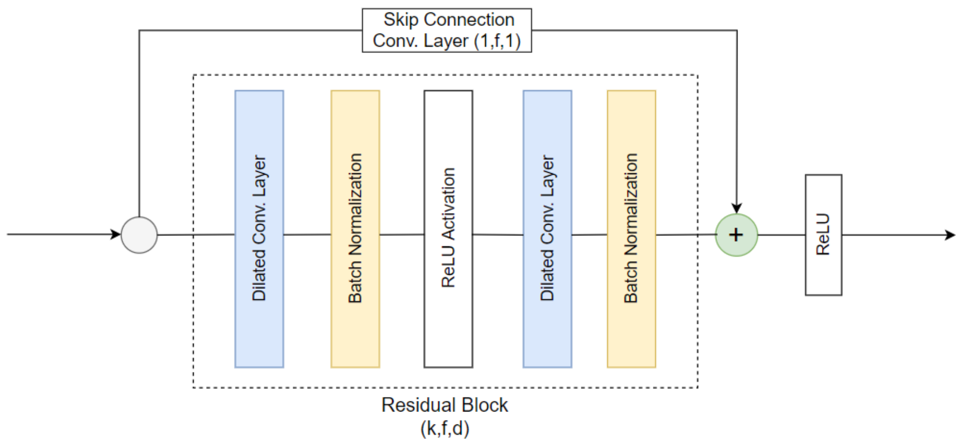


Figure 10: TCN residual block, containing two identical dilated convolutional layers with kernel size = k, no. convolutional filters = f, and dilation factor = d. The skip connection consists of a convolutional layer with k = 1, d = 2, and the same number of filter as the convolutional layers within the residual block, followed by an element-wise addition $\oplus$

[11]https://github.com/oneday88/deepTCN

- **LSTM**: The LSTM network contains two hidden LSTM layers, followed by a fully-connected output layer used to map the output from the LSTM layers into the actual quantile forecasts. When stacking several LSTM layers, the output from the first LSTM layer is used as input to the next, as illustrated in Fig. 11. Both LSTM layers in the network contain an equal number of units. Stacking several LSTM layers makes the network deeper, and the field of deep learning is built around the idea of deeper network being more effective at representing some functions compared to shallower networks (Bengio, 2009). The idea of stacking several recurrent levels is to encourage each recurrent layer to operate at different timescales (Pascanu, Gulcehre, Cho, & Bengio, 2013), improving the network performance for more challenging sequence prediction problems.

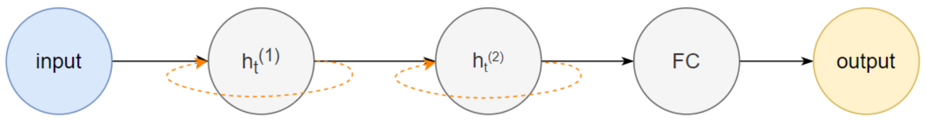


Figure 11: Two-layered LSTM network with a fully-connected layer (FC) to output the predictions.

Similarly to the EnbPI algorithm, the EnCQR method uses homogeneous ensembles, i.e. the member learners are identical and the diversity between them is achieved by training the individual networks on subsets of the original training dataset. An illustration of an ensemble of LSTM networks is given in Fig. 12. However, the way the subsets are sampled differ for the two methods: the EnbPI algorithm uses bootstrap estimators trained on random subsamples of the training dataset that disregard the ordering of the data points, whereas the ensemble learners in the EnCQR algorithm are trained using disjoint[12] subsets in the form of sequences. The first difference is that the bootstrap samples of EnbPI are drawn with replacement, while the subsets of EnCQR have no data points in common. A second difference is that the underlying regression algorithms considered in this section both require that the subsets used to train ensemble member learners maintain the sequential ordering of the training data in order to learn the temporal structures within the sequence. Therefore, the ensemble subsets are created by splitting the original training dataset into shorter sequences. Subsampling, as described in Section 7.6, creates multisets from the original data by extracting subsets with size smaller than the original dataset, *without* replacement. The subsampling of the training data is further explained in Section 14.1.

[12]The ensemble subsets must be disjoint to ensure that the leave-one-out estimations made by the ensemble learners are independent. If the subsets are not disjoint, the training residuals calculated by the leave-one-out estimators are not out-of-sample, and are then unrepresentative of the actual residuals.

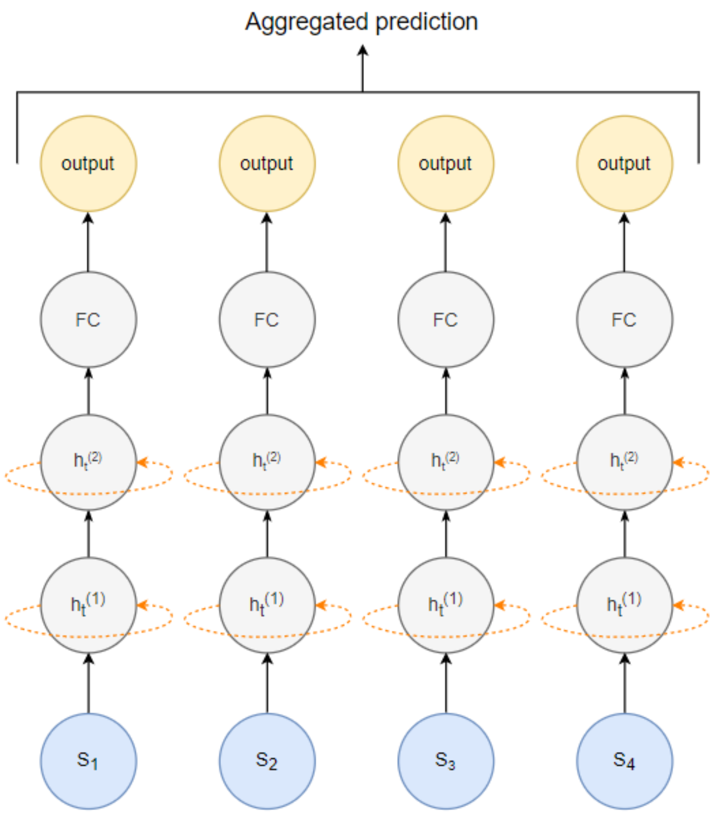


Figure 12: A ensemble of four two-layered LSTM networks, each trained on a subset $S_i, i = 1, .., 4$ of the original training data.

Part IV / Experiments

This part of the thesis describes the experiments conducted to assess the performance of the proposed model. The EnCQR model is tested on both univariate and multivariate electricity load time series, and its performance is compared with two other neural network-based models and a seasonal ARIMA model. The proposed model uses an ensemble of neural networks performing deep quantile regression combined with conformal prediction to construct probabilistic forecasts, resulting in a complex model architecture. Comparing the model performance with both similar and simpler models assesses the model capability and whether the increased complexity is rewarding. The first neural network-based reference model is the EnbPI algorithm, due to its close relationship with the proposed model. The other neural network-based reference model is a single neural network performing deep quantile regression, denoted QRNN. Both neural network-based reference models are implemented using the same two regression algorithms as the proposed model, described in Section 12.2.1. A seasonal ARIMA model is chosen as the baseline statistical reference model, as the ARIMA models represent one of the most traditionally used models within the field of time series forecasting.

In Section 13 the two datasets used in the experiments are presented, along with the steps performed in the preprocessing of the time series. In Section 14 the architectures and implementation details of the models used in the experiments are described. The evaluation metrics used to assess the models' performance are described in Section 15. Finally, Section 16 presents and discusses the experimental results.

13 Datasets

In this section, two different real-world electricity load datasets are presented. For each dataset, a pre-analysis is performed to study the variance and detect possible trends and seasonalities within time series data, which is further used to determine the necessary preprocessing to be performed. Data preprocessing includes, if necessary, deseasonalizing, detrending, and normalizing the data. If data preprocessing is performed, an appropriate post-processing must be applied to transform the data back to its original format when evaluating the results.

As described in section 6.5.1, statistical forecasting models such as ARIMA assume stationary, and the data must therefore be examined to identify if this assumption is met. If the results of the examination shows that the time series is in fact non-stationary, data preprocessing must be performed before fitting the models to the data.

13.1 Portugal Dataset - Univariate

The Portugal dataset[13] describes the electricity consumption of 370 customers located in Portugal. The data are recorded per 15 minutes from 2011 to 2014 for each customer, and each column in the datasets represent one customer, i.e. one individual household. Some customers were created after 2011. In these cases, the consumption before creation is considered zero. Due to daylight saving time, which occurs every year in March and October, time changes respectively. At time change day in March, which only has 23 hours, the values between 1:00 am and 2:00 am are zero for all points, and in October time change day, which has 25 hours, the values between 1:00 am and 2:00 am aggregate the consumption of two hours. The values of the time series are converted into hourly consumption by aggregating records of the same hour, i.e. aggregating blocks of 4 measurements. Data from the last three years, i.e. 2012, 2013, 2014, is used, and each time series consist of 26,304 measurements.

To conduct the experiments, five of the 370 individual time series are randomly selected, all plotted in Fig. 13. The yearly seasonality of the individual time series is more pronounced for some, e.g. station 77 and 27, whereas in station 250 and 90 less so. Table 2 presents descriptive statistics of the five selected time series, showing that the consumption level varies significantly between each individual household, and data preprocessing is therefore performed individually for each time series. The data analysis and preprocessing operations are shown in detail here only for the time series associated with station 250, but the methodology is applied identically for all the other time series.

Table 2: Descriptive statistics for the five time series used for evaluation. The electricity consumption for each time series is given in kiloWatt (kW).

Station	250	77	50	90	27
length	26304	26304	26304	26304	26304
mean	307.18	160.98	221.29	553.33	268.76
std	89.13	43.83	62.67	148.51	85.44
min	41.18	32.97	46.18	34.02	73.15
max	531.68	436.42	529.31	1170.780	676.61

A load profile is a graph showing the changes in electricity load over a specific time. As described in Section 3, multiple factors influence electricity consumption. The shape of the load profile varies according to factors such as customer type and activities, time and date, and climatic conditions. Since the time series in this dataset represent hourly electricity consumption of individual households, we expect the load profile to show a clear daily cycle, where the load peaks during the day and reaches a minimum during the night when customer activity is low.

[13]avaliable at `https://archive.ics.uci.edu/ml/datasets/ElectricityLoadDiagrams20112014`

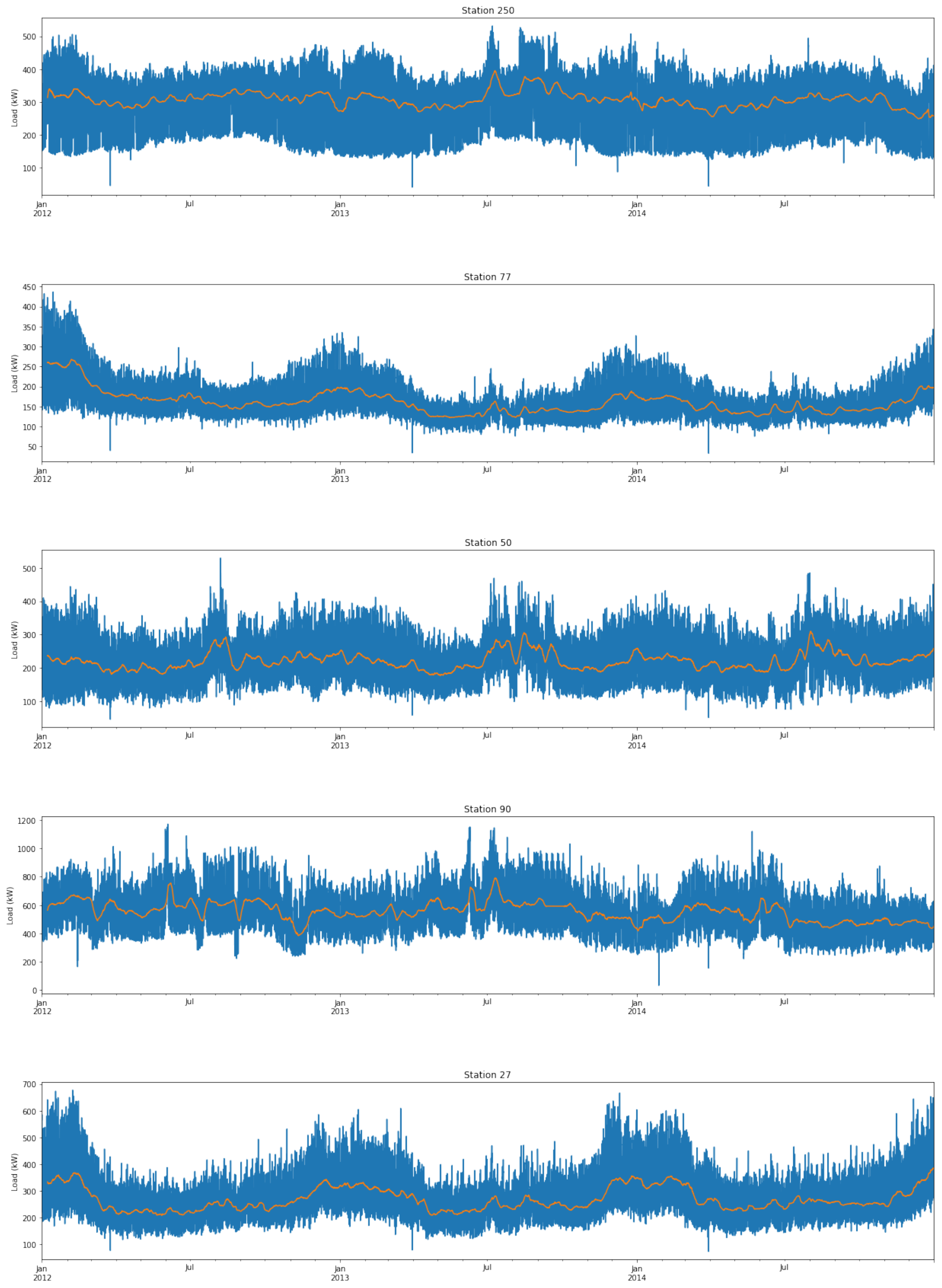


Figure 13: Electricity load profiles for all stations. The orange line corresponds to the weekly rolling average mean consumption (168 hours). None of the time series show a clear visible trend, but a seasonality corresponding to the yearly seasons are visible for some.

Figure 14 shows the weekly load profile of 1-7 April 2012 for station 250. As expected, the daily cycle is clear, where the consumption is mainly concentrated in the central hours of the day, also observed in the boxplot displayed in Fig. 15c. Boxplots presents a robust summary of the distribution of a dataset using five components: the minimum (0th percentile), the first quartile (25th percentile), the third quantile (75th percentile), and the maximum (100th percentile), as well as possible outliers (McGill, Tukey, & Larsen, 1978). The box extends from the first quartile to the third quartile values of the data, and the median is illustrated as a line, dividing the box into two parts. The interquartile range (IQR) is the difference between 75th and 25th percentiles, and is used as the measure of statistical dispersion. The whiskers extend from the edges of box to show the range of the data, i.e. the minimum and maximum value, extending no more than $1.5 \times \text{IQR}$ from the edges of the box. Points that falls outside the whiskers are termed outliers, plotted as separate dots.

Additionally, Figure 15c shows that the hourly variability is larger during the central hours, indicating heteroscedasticity. The yearly and monthly variability are shown in Fig. 15a and 15b, respectively. The former being nearly constant, showing no obvious trend, whereas a clear seasonal variability can be seen in the monthly consumption, peaking at the summer and winter months. The monthly variability is expected since the load profile is highly affected by temperature; the consumption is high in the winter and summer months due to the usage of heating and cooling, respectively.

The seasonality of the time series is analysed using the autocorrelation function, plotted in Fig. 16. Usually, for a stationary time series, the ACF plot decays rapidly from the initial value of unity at zero lag (Adebiyi, Adewumi, & Ayo, 2014), whereas for a non-stationary time series, the ACF dies out gradually over time. The ACF plot for Station 250 show a slowly decaying correlation, and a clear seasonal pattern every 24 hours, corresponding to the daily variations, indicating non-stationary. To further investigate the seasonality of the time series, the daily cycle is filtered out using a seasonal differencing with lag 24. The orange line in Fig. 16 represents the result after the differentiation. When removing the daily seasonal variability, a previously hidden pattern is revealed; the orange line peaks around lag 168, corresponding to a weekly cycle. The negative peak at lag 24 is introduced by the differentiation. The ACF plot of the seasonally differentiated time series shows that the autocorrelation goes to zero after few lags. The repeating pattern and exponential decay at each 24th lag in the PACF plot in Fig. 16c indicates the seasonality of the time series. The ACF and PACF plots, together with the AIC, are used to determine the SARIMA model orders, described later in Section 14.2.

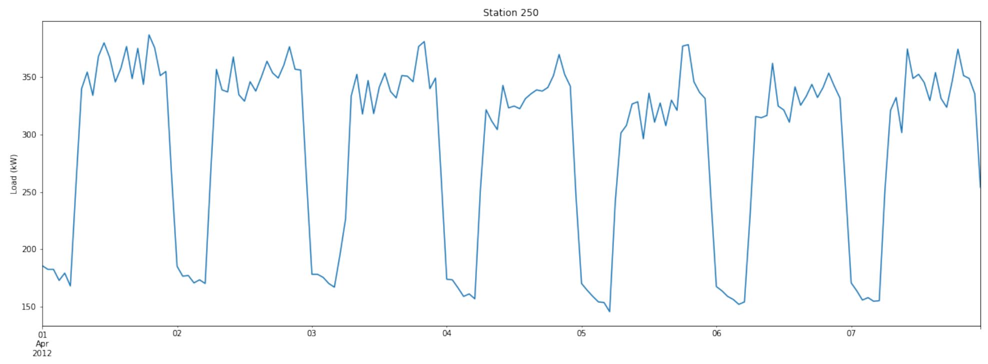
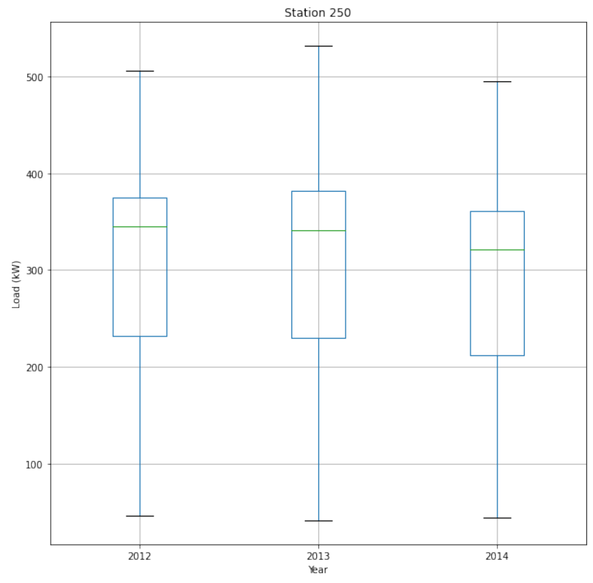
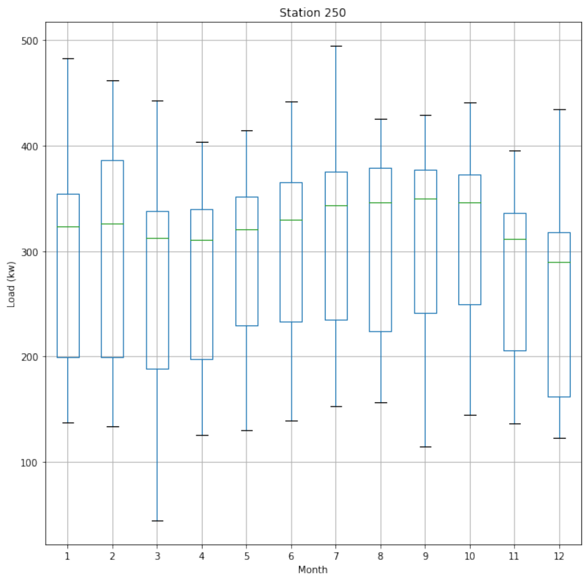


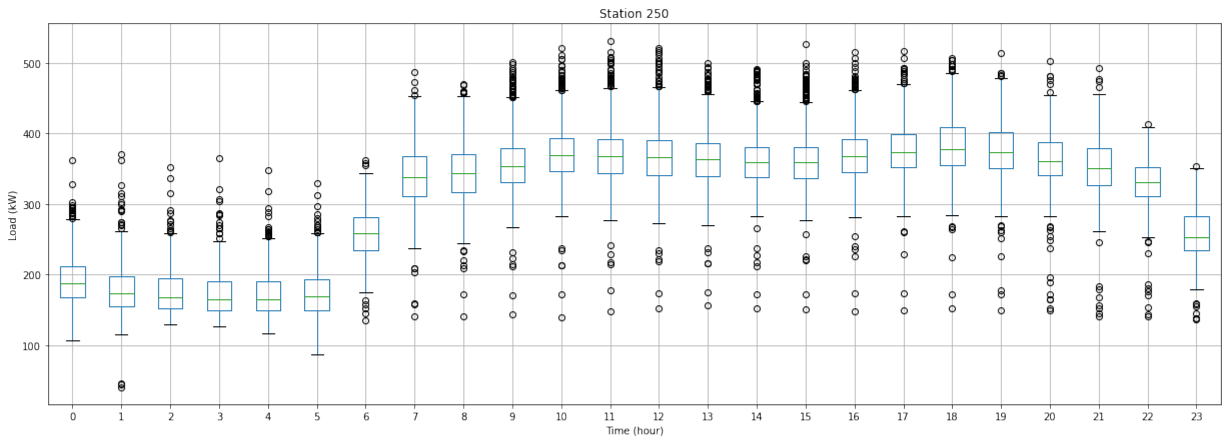
Figure 14: The hourly load profile of the electricity consumption of station 250, registered over one week.



(a) Yearly variability

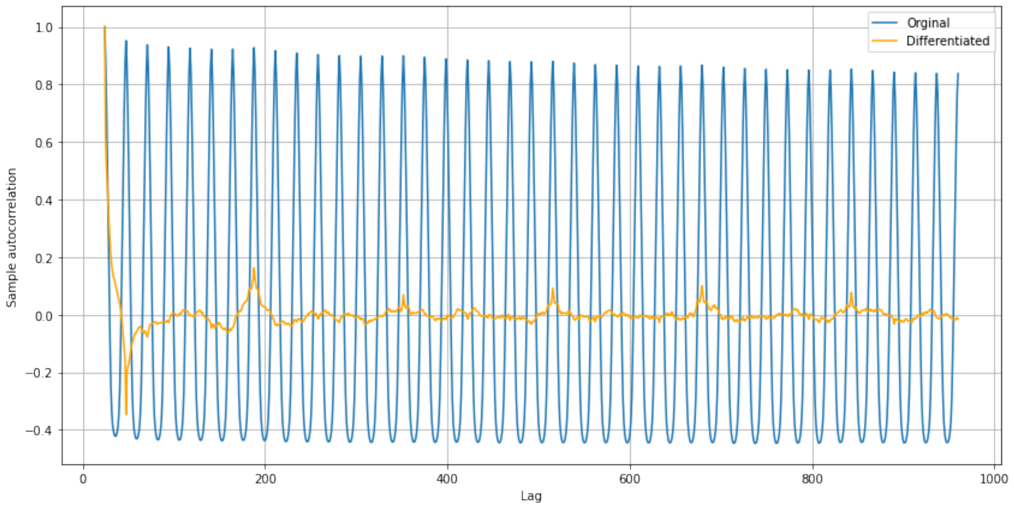


(b) Monthly variability

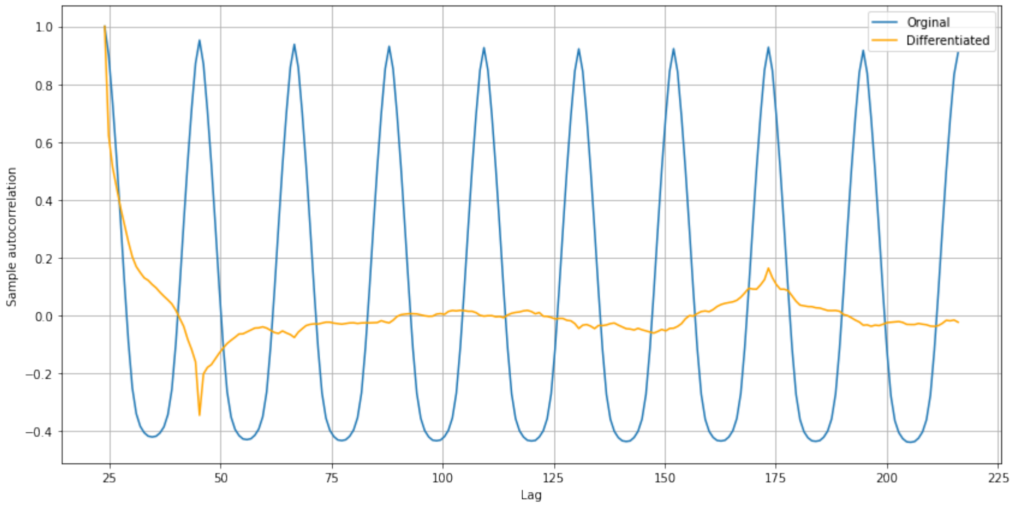


(c) hourly variability

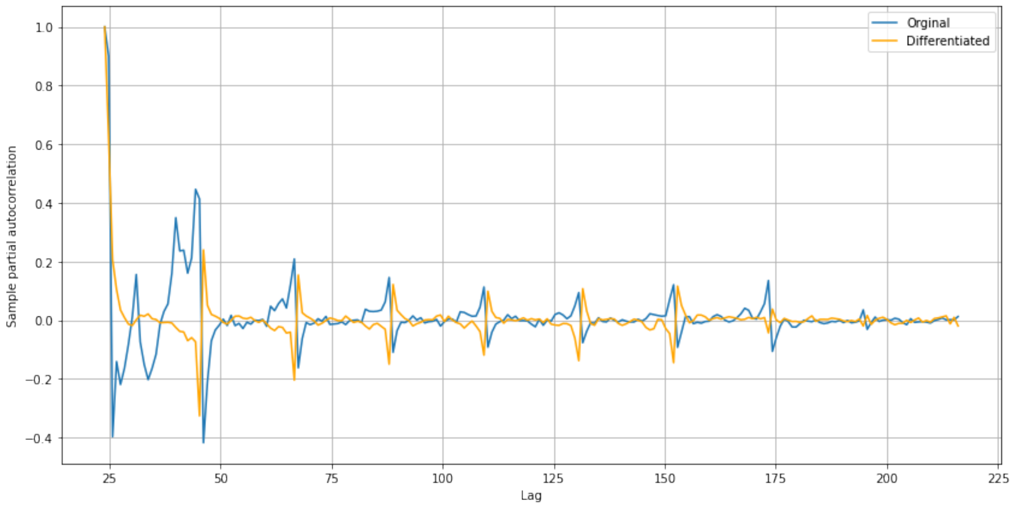
Figure 15: Yearly (a), monthly (b), and (c) hourly variability in electricity load for all years. Figure (a) shows no clear rising or sinking trend, and the yearly variability is almost constant. Figure (b) shows a seasonal pattern through the year; the median load is highest in the winter and summer months. The daily cycle in (c) shows that the consumption is highest in the central hours of the day, where the variability also is largest.



(a) Autocorrelation (940 lags)



(b) Autocorrelation (216 lags)



(c) Partial autocorrelation (216 lags)

Figure 16: The ACF and PACF of station 250 before (blue line) and after (orange line) a seasonal differentiation at lag 24. The original time series shows a strong seasonal pattern at lag 24, which corresponds to a daily cycle. After seasonal differencing, a new seasonal pattern is revealed at lag 168, corresponding to the weekly cycle.

13.1.1 Data Preprocessing

The time series are partitioned into three disjoint datasets; training, test, and validation data, where the training data represent the first year, the following year is used for validation, and the last year is used as test data. The data partitioning is based on Fig. 15a, where such a split is considered reasonable since the yearly variability is more or less the same across the three years. For all time series, the training dataset consist of 8784 measurements, whereas the validation and test datasets consist of 8760 time steps. The difference is 24 measurements, corresponding to one day, since the year 2012 had 366 days. All time series are independently normalized by scaling the values to lie in the interval [0,1], i.e. min-max normalization using the MinMaxScaler from the *sklearn* library in Python. The scaler is fitted on the training data and then used to normalize validation and test sets.

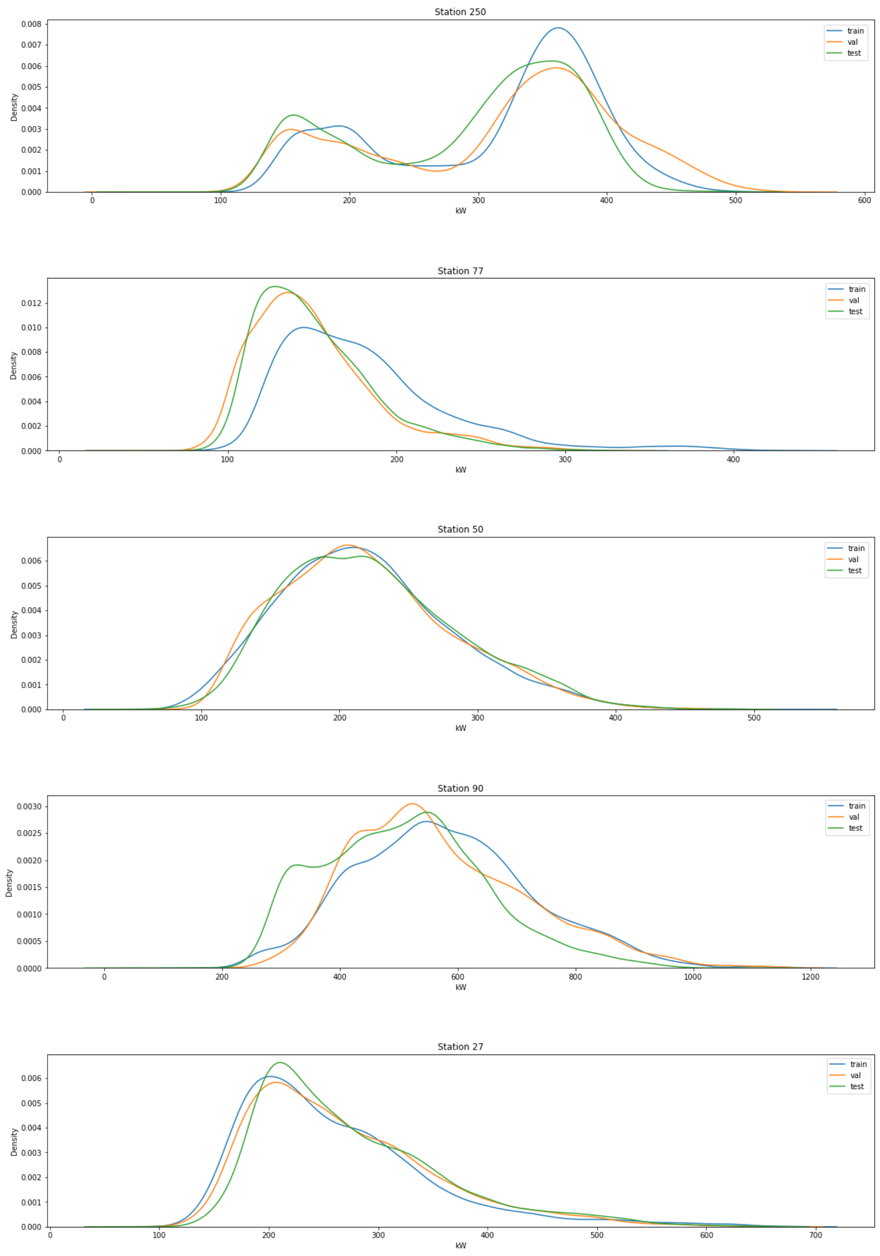


Figure 17: Kernel density estimation of the probability density functions of the train, validation, and test datasets for all stations time series

Figure 17 shows kernel density estimates[14] of the train, validation, and test datasets for the five time series before normalization. To construct a well-performing model, it is essential that the data used to train and evaluate the model originate from the same target distribution. The validation dataset is used to tune the model's hyperparameters, hence, it indirectly affects the model. Histograms or kernel density estimates can be used to assess the data distributions and, from Fig. 17, it is clear for all time series that the train, validation, and test datasets are reasonably similar in terms of the distribution of the observations within each set. Variations in the probability density functions of the three datasets cannot be avoided, as the variations reflects natural variations between the years, but the homogeneity between the three datasets are sufficiently similar for the purpose of the analysis.

In addition to splitting the data into train, validation, and test data, the time series must be transformed into input-output pairs in order to train the neural network-based models. The method of transforming a time series into a supervised learning problem is described in Section 6.3. In the experiments, the length of the input samples is determined by analyzing the seasonality of the time series; After a seasonal differentiation at lag 24, all time series show a second seasonal pattern with a period of 168 lags, hence, the size of the input samples is set to 168. The output sequence consists of 24 measurements, since the models are designed to predict one day ahead using the previous seven days. Consequently, the first week of the train, validation, and test datasets cannot be predicted since historical data for these days is unavailable. The predictive setup is illustrated in Figure 18.

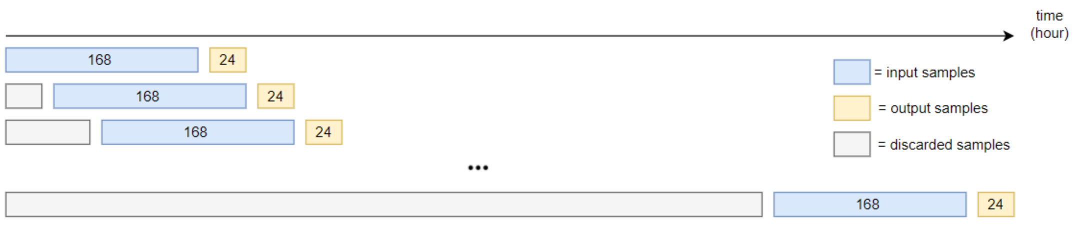


Figure 18: Predictive setup using sample pairs with input and output size of 168 and 24, respectively. Total sample length = 192 hours.

A second transformation is performed to convert the two-dimensional structure of the supervised learning data into a three-dimensional structure required by the convolutional and recurrent networks. The structure of the input data must be on the form $[\text{samples}, \text{time steps}, \text{features}]$, whereas the structure of the time series data in the example above is $[\text{samples}, \text{time steps}]$. The additional dimension, i.e. the *feature* dimension, refers to the number of variables recorded at each time step in the time series. Since only the load is monitored, the time series is univariate and the features dimension is equal to one.

[14]Kernel density estimation (KDE) is a non-parametric method to estimate the probability density function of a random variable (Terrell & Scott, 1992)

13.1.2 Data Complexity

To assess the complexity of the forecasting problem, it is possible to evaluate the error achieved by baseline models. Baseline models are simple, computationally fast, and often deterministic models used to establish a baseline performance level for comparison with other more advanced models (Brownlee, 2018). A commonly used baseline model is the naive approach, assuming that the next day is similar to the previous day, which construct forecasts for $(t + i), i = 1, \ldots, 24$ using the value at the same times in the previous day, $(t + i - 24), i = 1, \ldots, 24$. This approach can be extended, assuming that the days in the next week are similar to the previous week, i.e. constructing forecasts for $(t + i), i = 1, \ldots, 24$ using the values at the same times in the previous week, $(t+i-168), i = 1, \ldots, 24$. Since electricity load not only has a daily pattern, but also a strong weekly pattern, the latter approach is utilized.

By calculating the MSE of the naive forecasts, an assessment of the data complexity can be obtained. The MSE is a measure of the quality of the forecasts, where values closer to zero indicates more accurate predictions. Consequently, if the MSE of the naive method is high, the forecasting problem can be assumed to be of increased difficulty. The MSE for all time series in the Elvia dataset is presented in Table 3, showing that the MSE is lowest for Station 250 and Station 77, and highest for Station 50, but the deviation between the time series is relatively small.

Table 3: MSE of the naive predictions of the normalized full dataset for all five time series.

Time series	**Mean squared error**
Station 250	0.00430
Station 77	0.00266
Station 50	0.00821
Station 90	0.00596
Station 27	0.00410

Figure 19 shows the naive forecasts for a period of two weeks for all time series in the dataset. The naive method seems to be predicting with reasonable accuracy, since the variations between the subsequent weeks are minor. Note that the figure only show the results from a small part of the test data, and is only meant as an illustration. For a holistic assessment, the MSE scores must be considered.

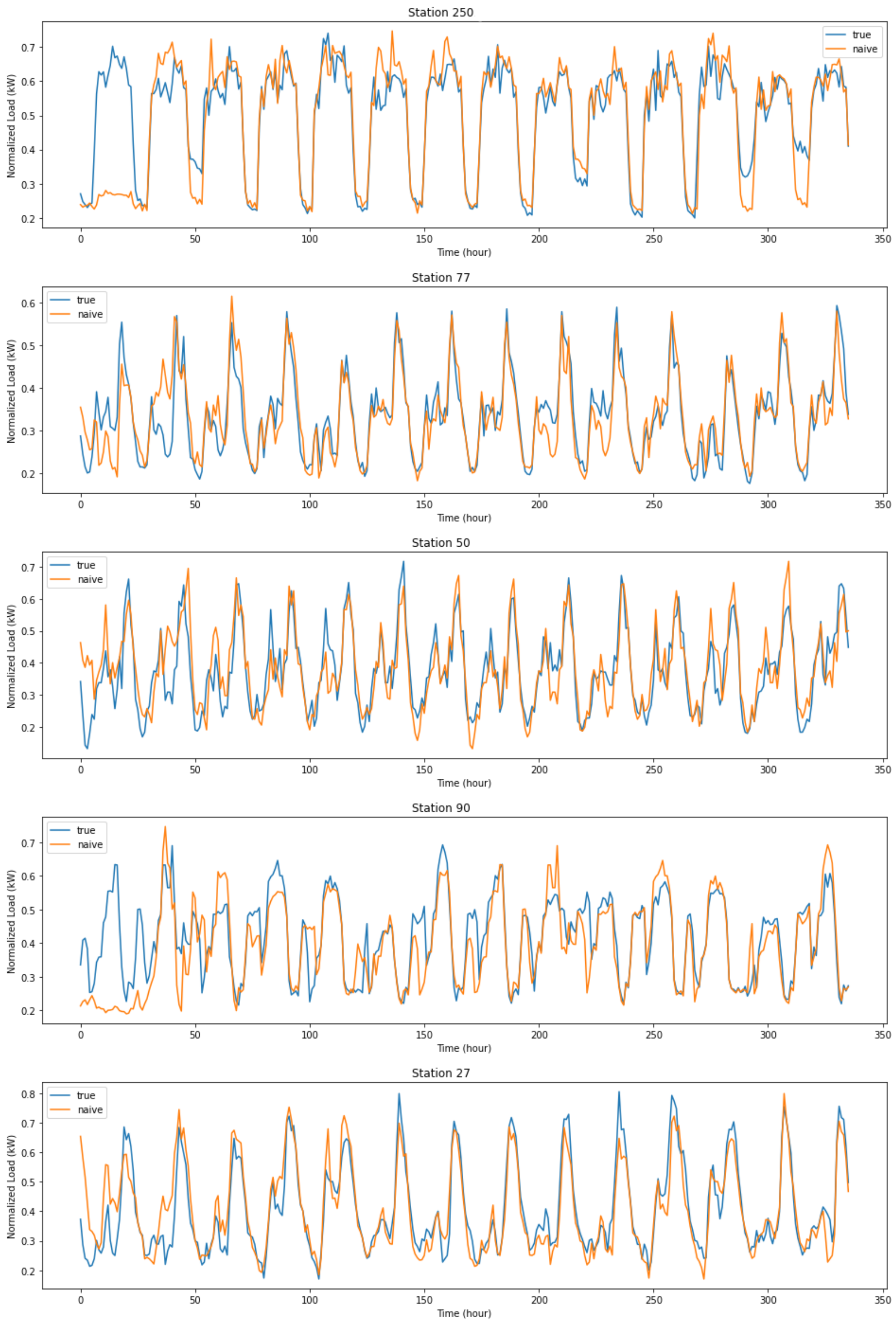


Figure 19: Naive forecasts for a period of two weeks for all five time series from the Portugal dataset. The blue line corresponds to the true observations, and the orange line is the naive predictions.

13.2 Elvia Dataset - Multivariate

The *Elvia* dataset is shared by Elvia AS, a distribution system operator that operates distribution grids in the counties of Oslo, Viken and Innlandet, Norway. The dataset describes the electricity consumption at a secondary substation level for 4749 stations, and includes a time series of historical temperature forecasts for these areas. The stations consists of data from three different user categories; cabin, industry and household. All secondary substations have 5 or more connected end-users, and time series data is the sum of the consumption of all connected end-users. The data is recorded per hour from 1 June 2018 to 1 September 2020, resulting in 19,962 measurements per time series. The energy consumption is reported in kWh per hour.

In the experiments, one time series from each user category is randomly selected. The data analysis is shown for all three time series, since they, unlike the time series in the first electricity dataset, come from different end users and are therefore expected to have different characteristics.

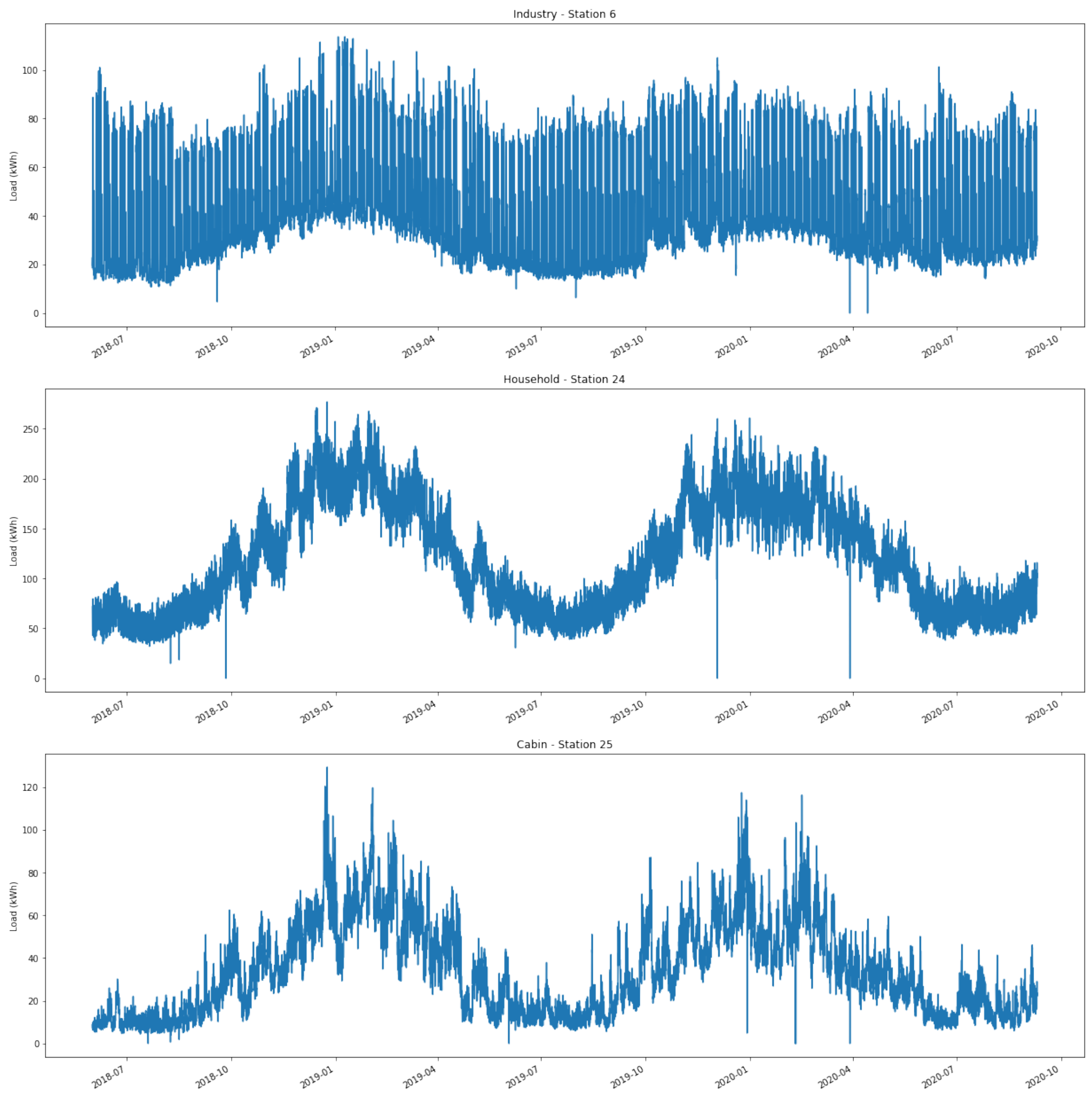
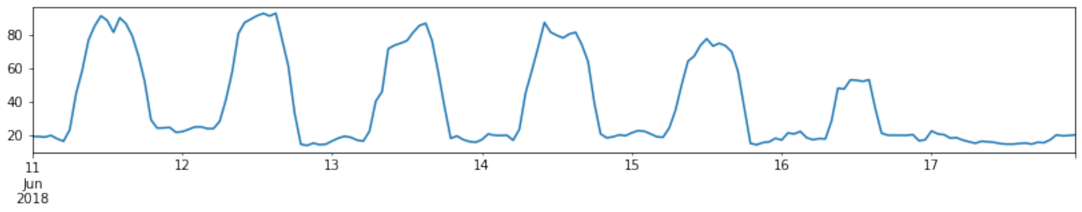


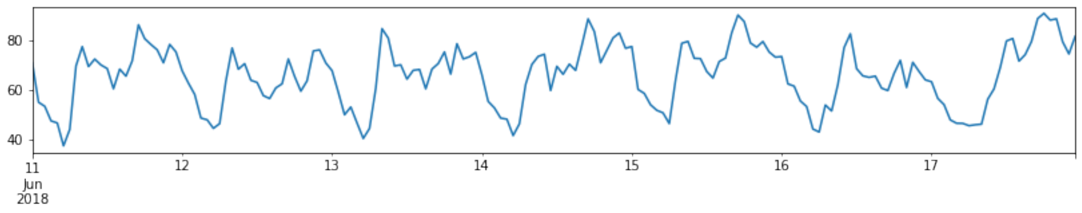
Figure 20: Electricity load profiles for all three end users. All time series show a clear seasonality, where the consumption peaks during the winter months.

Figure 20 shows that all the time series have a clear seasonal pattern, where the consumption peaks in the winter months and significantly reduces during the summer months. Such seasonal patterns are expected for electricity consumers in Norway, where the use of heating significantly increases during the winter, whereas during the summer months, practically no electricity is used for neither heating nor cooling.

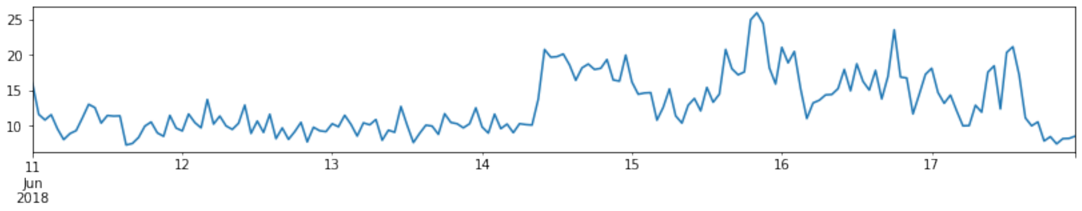
The weekly load pattern for the different end-users is expected to differ, since the primary consumption for industry is usually concentrated on the weekdays when people are working. Contrarily, most of the consumption for cabins is expected to be concentrated around weekends. However, this varies significantly with public holidays. For households, the consumption is expected to be roughly the same for all weekdays, with a clear daily pattern. Figure 21 illustrates precisely this; the industry load peaks during the central hours and reduces during late evening, early morning, and weekends, whereas the cabin load profile is low on weekdays and rises during the weekends. The household load profile shows a clear daily pattern, where the consumption peaks during the central hours of the day.



(a) Industry - station 10



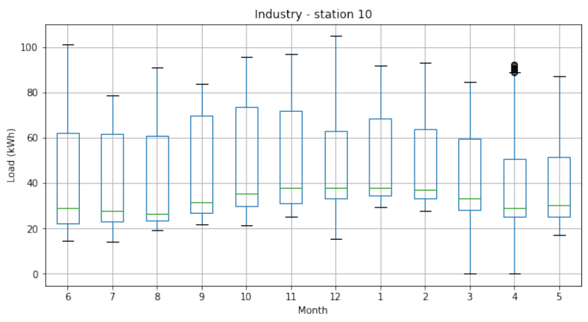
(b) Household - station 24



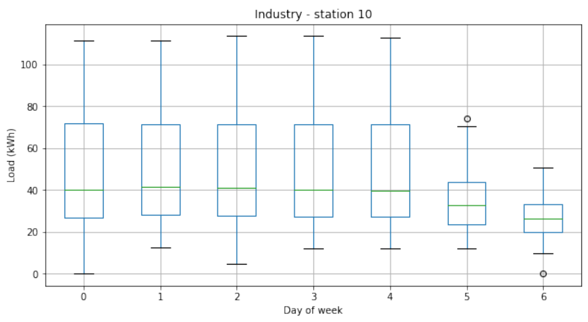
(c) Cabin - station 25

Figure 21: The hourly load profile of the electricity consumption of the Industry, Household and Cabin time series, registered over one week (Monday 11 June 2018 - Sunday 17 June 2018).

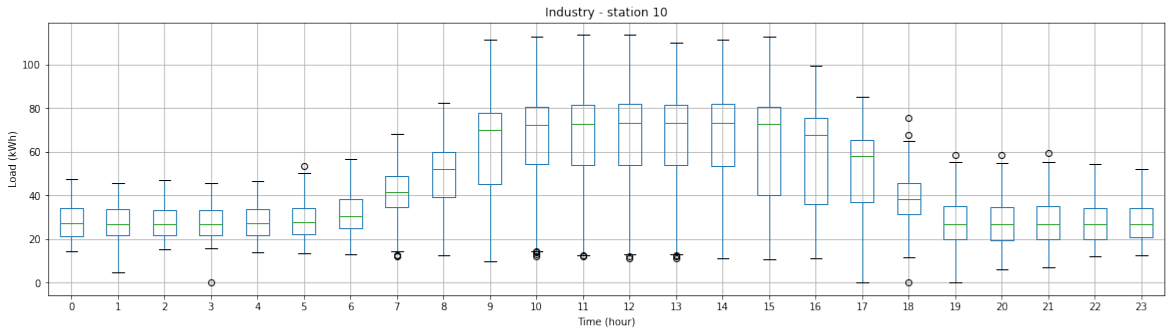
The monthly, daily, and hourly variability for the Industry, Household, and Cabin time series is analyzed in detail in Figures 22, 23, and 24. All figures show that the hourly variability change for different times of the day, indicating heteroscedasticity.



(a) Monthly variability

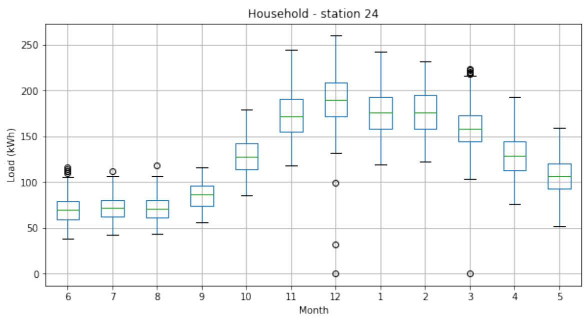


(b) Daily variability

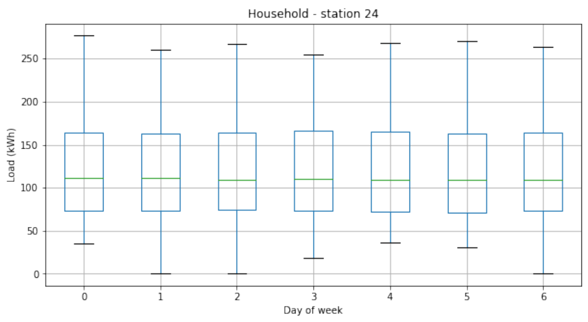


(c) Hourly variability

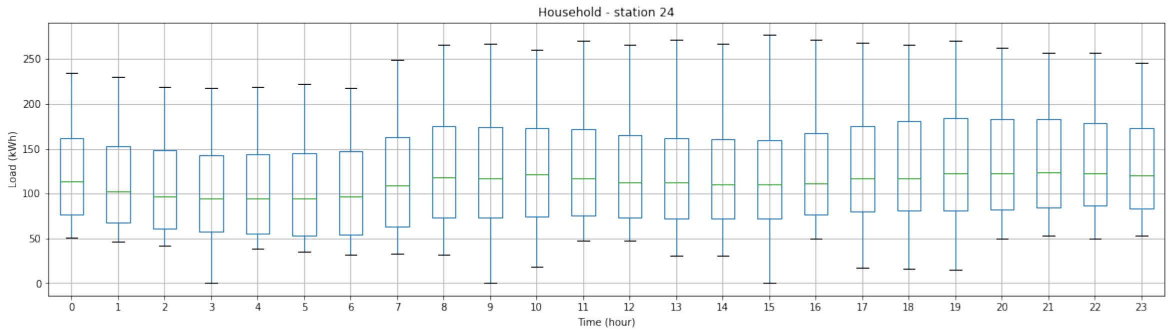
Figure 22: Electricity load monthly, daily, and hourly variability for the Industry time series. Figure (a) shows that the consumption peaks in the winter months, where the variability also is the largest. The consumption drastically sinks at the weekend, and at the earliest and latest hours of the day, corresponding to when the industry is closed.



(a) Monthly variability

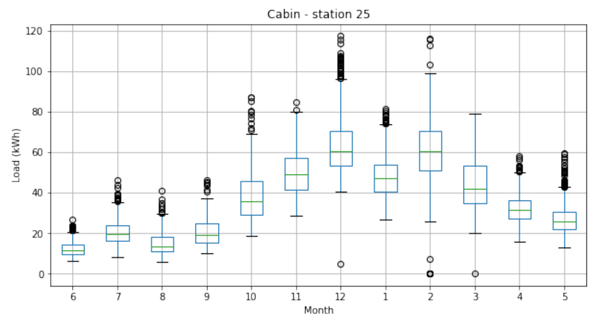


(b) Daily variability

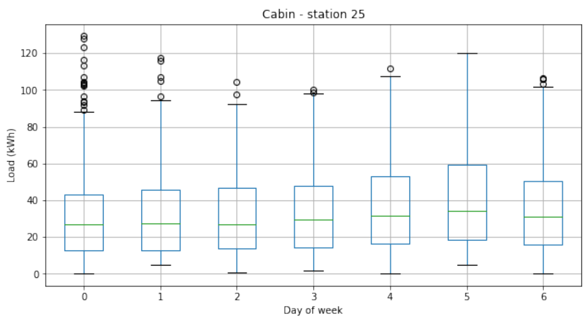


(c) Hourly variability

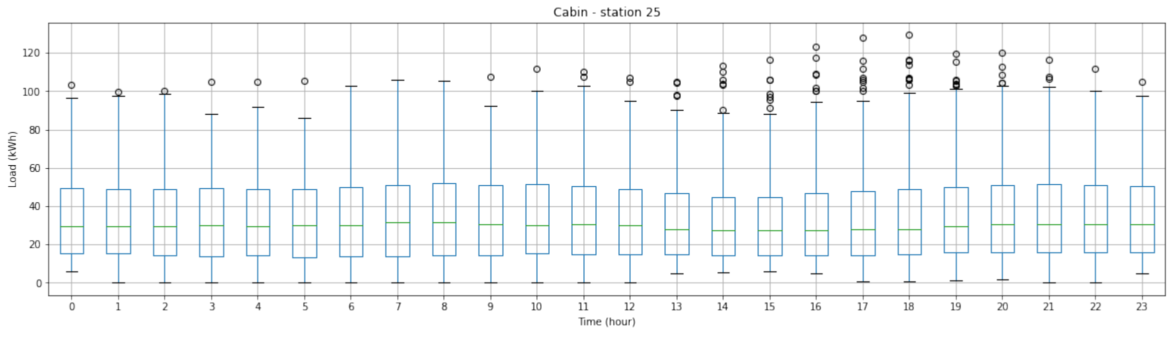
Figure 23: Electricity load monthly, daily, and hourly variability for the Household time series. Figure (a) shows that the consumption peaks in the winter months. The median and interquartile range of the consumption show small differences for the different days of the week, as expected.



(a) Monthly variability



(b) Daily variability

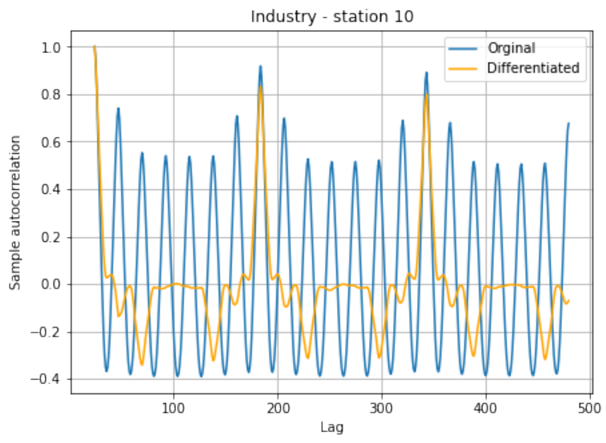


(c) Hourly variability

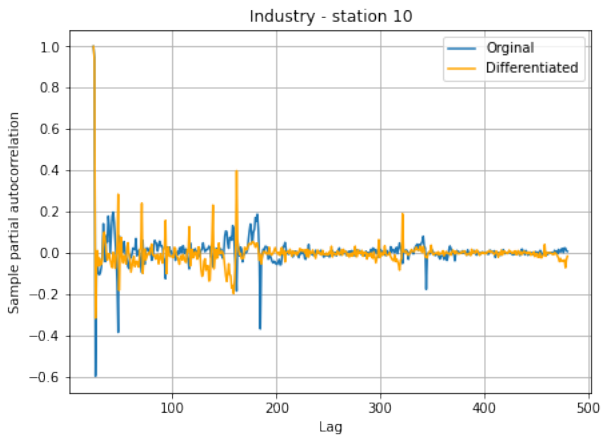
Figure 24: Electricity load monthly, daily, and hourly variability for the Cabin time series. Figure (a) shows that the consumption peaks at the winter months, where the variability also is the largest. All plots show a significant amount of outliers, which is to be expected, since the electricity consumption at cabins varies significantly with weather conditions and holidays.

The seasonality of the time series is further analyzed using the ACF and PACF plots in Fig. 25. The ACF plot for both the Industry and Household time series shows a clear seasonal pattern, with a cycle of 24 hours corresponding to the daily variations. After a differencing at lag 24, a strong weekly pattern for the Industry time series is revealed, narrowly peaking around lags corresponding to one week. This weekly pattern is also present for the Household time series but is significantly weaker, indicating that the difference in electricity consumption between the days in a week is much smaller than for the Industry time series.

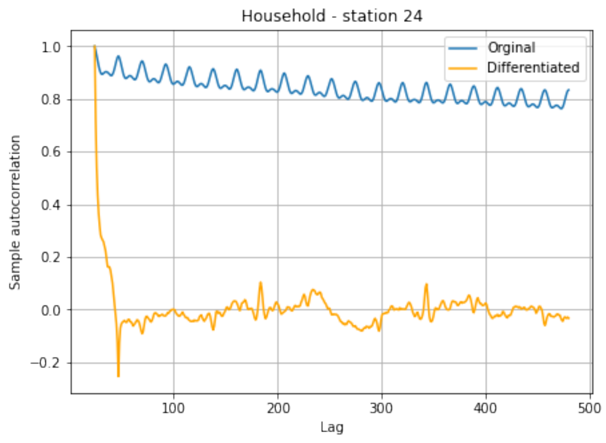
Cabins are mostly used during the weekends, and the consumption during these days is often significantly higher than for other days of the week. A seasonal differentiation at lag 24 reveals a second seasonal cycle, representing a broader cycle compared to the industry time series. This seasonality appears most likely because the weekends are highly correlated with each other but less similar to the other days of the week. The differences in the ACF and PACF plots for the three time series indicate the distinct characteristics of each end-user, agreeing with the statements made above.



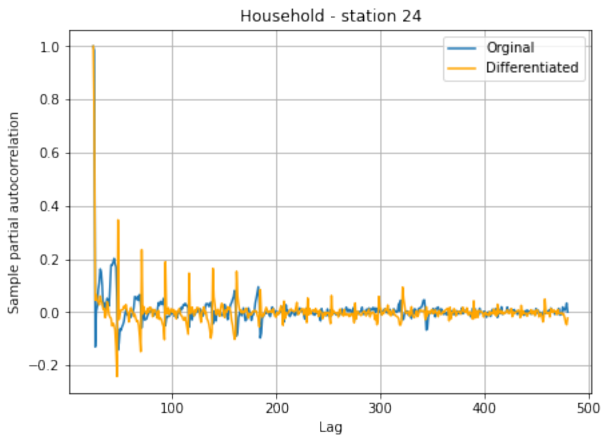
(a) ACF - Industry



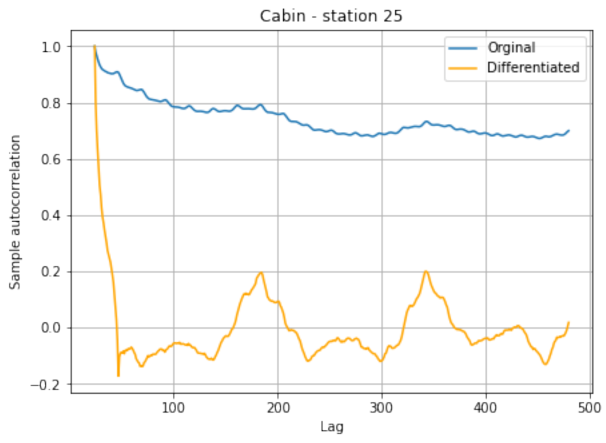
(b) PACF - Industry



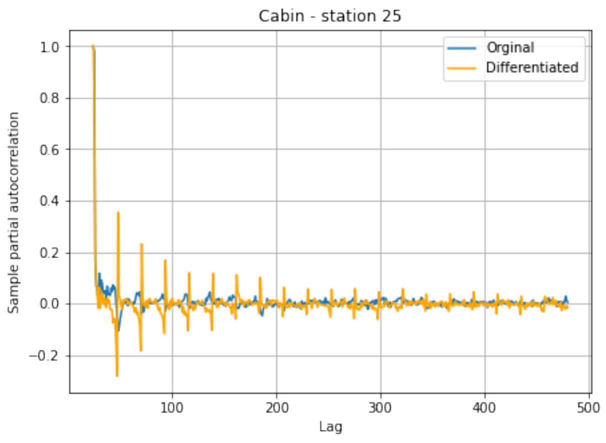
(c) ACF - Household



(d) PACF - Household



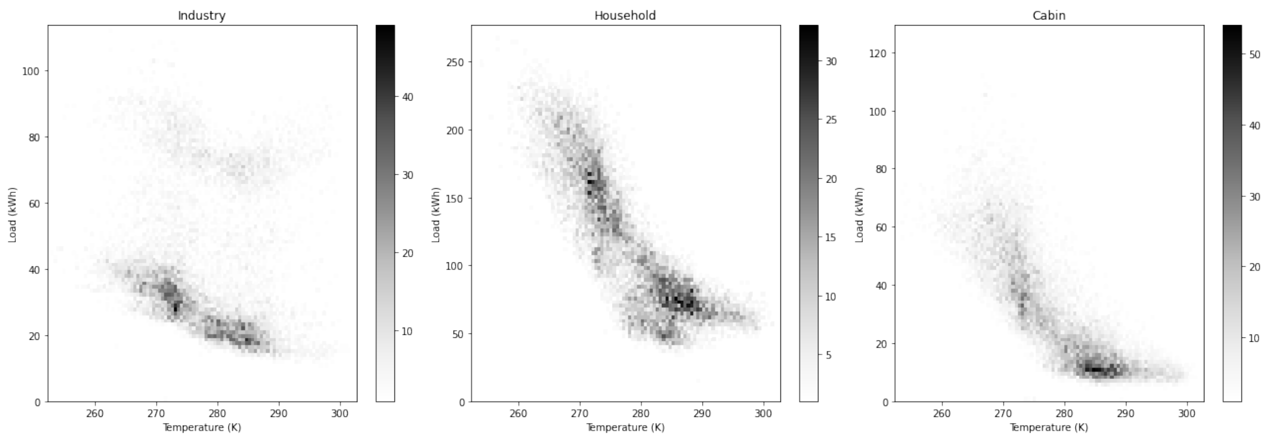
(e) ACF - Cabin



(f) PACF - Cabin

Figure 25: The ACF and PACF of the Industry, Household and Cabin time series (blue line) and after (orange line) a seasonal differentiation at lag 24.

To improve the accuracy of electricity load forecasts, temperature is commonly included in forecasting models as an exogenous variable, since electricity consumption generally is strongly related with temperature. The relationship between load and temperature is known to be non-linear, where both high and low temperatures can cause an increase in electricity consumption. The temperature time series included in the Elvia dataset contains temperature forecasts from a single site in the area where the substations are located, resulting in the substations being positioned with varying distances from the chosen site. Therefore, the temperature forecasts can be less accurate for the substations located furthest away. To assess the association between consumption and temperature for the three different end-uses, the correlation between load and temperature is analyzed using two-dimensional histograms, depicted in Figure 26.



(a) Industry (b) Household (c) Cabin

Figure 26: Two-dimensional histogram of electricity load and temperature for all end-users. Darker areas represent more populated bins and the bar on the right indicates the number of elements in each bin. The plots show a strong negative correlation for the household and cabin time series, resulting from the increased use of heating at lower temperatures.

The two-dimensional histogram estimates the joint distribution of load and temperature, and a strong negative correlation between the Household and Cabin time series and the temperature can be seen in the figures, indicating that electricity consumption is highest at lower temperatures. The results from the histograms agree with the discussion above, as the electricity consumption of customers located in Norway significantly increases during the winter months due to heating and decreases during the summer months, since cooling devices are seldom used. The correlation between temperature and electricity load for the Industry time series is not as strong as for the two other categories. The most probable reason is that industrial consumption is more affected by calendar effects, and due to the nature of the energy use the weekly consumption is more constant throughout the year.

From the analysis of the histograms above, one can expect the accuracy of the forecasting models for both the Household and Cabin time series to be improved by the additional information gained by including the temperature. However, the Industry time series seems less affected by temperature, and a lower increment in the prediction accuracy is expected to be gained by including temperature as additional input to the forecasting models. Regardless, the temperature forecasts are included in the models for all three time series.

13.2.1 Data Preprocessing

Following the same approach as in the preprocessing of the Portugal dataset, the time series are partitioned into train, validation, and test datasets, and transformed into input-output pairs with size 168 and 24, respectively. The temperature time series is partitioned into input and output pairs using the same sliding window as for the electricity load observations. The additional feature dimension in the three-dimensional data structure required by the convolutional and recurrent networks, is for the Elvia time series equal to two, since both the load and temperature is monitored (the time series are multivariate). The Elvia dataset contains measurements from 1 June 2018 to 1 September 2020, making the data partitioning not as intuitive as for the Portugal dataset.

To construct models that can produce satisfactory results when predicting future observations from any time of the year, the models must be trained on data from the whole year. To achieve this, the time series are shortened down to two full years, where the first year is used for training. To obtain similarly distributed validation and test data, odd and even months of the last year are used as validation and test data, respectively. For each month, the observations are partitioned into input-output samples, resulting in that the first week of each month in both the validation and test dataset cannot be predicted, since historical data for these days is unavailable. For the training dataset, historical data is unavailable only for the first week in the first month. The partitioning of the test and validation datasets are done similar to the illustration given in Fig. 29, where after the partitioning of the months, every other subset is combined to create the full test and validation datasets.

Figure 27 shows kernel density estimates of the train, validation and test datasets for the Elvia time series. From the figure, it is clear for all time series that the train, validation, and test datasets are reasonably similar in terms of the distribution of the observations within each set. For the Cabin time series, there is a slightly larger dissimilarity between the train and test dataset, compared to the two other time series. However, variations in the probability density functions of the three datasets cannot be avoided, as the variations reflects natural variations between the different months. For the Elvia time series, a increased dissimilarity between the three datasets can be expected, as the monthly variability, displayed in Fig. 22a, 23a, and 24a, is significant, but the homogeneity between the three datasets are sufficiently similar for the purpose of the analysis. Lastly, the three datasets are normalized using the MinMaxScaler fitted on the training data.

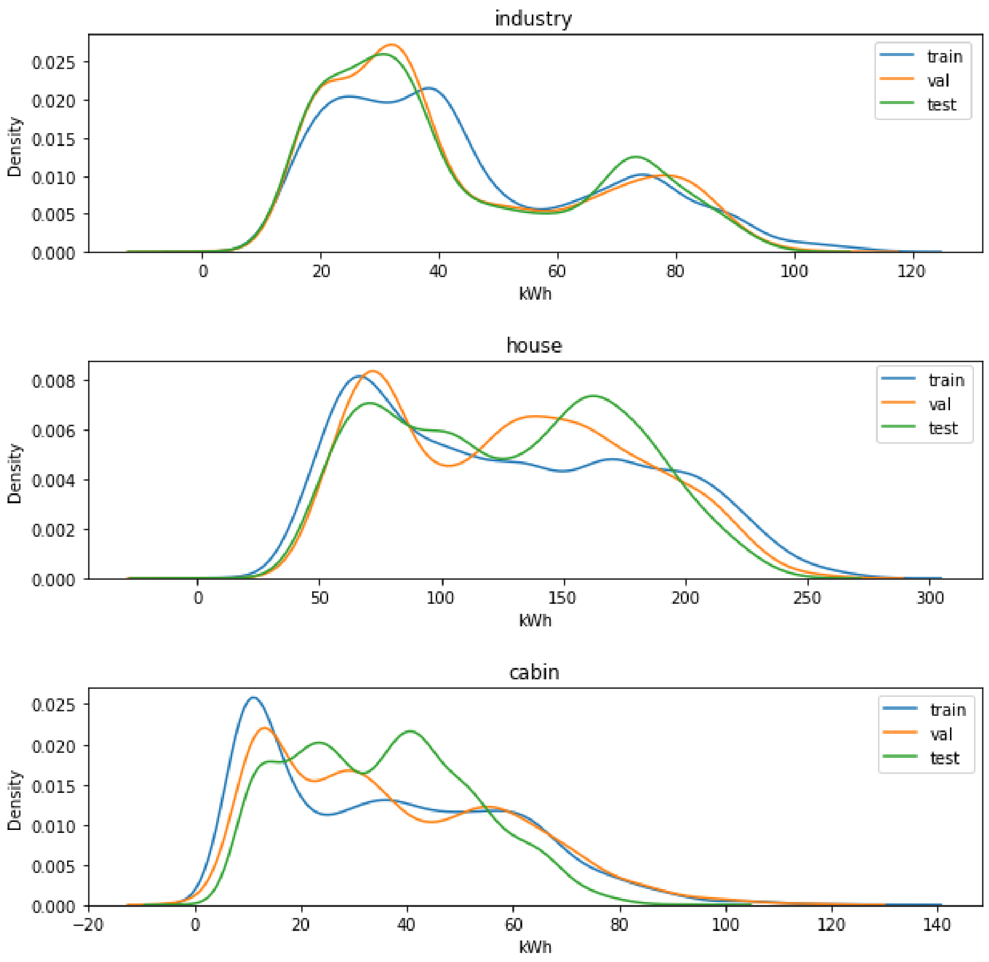


Figure 27: Kernel density estimation of the probability density functions of train, validation, and test datasets for all three end-users, showing that similarly distributed datasets.

13.2.2 Data Complexity

Similarly to the Portugal dataset, the complexity of the Elvia dataset is assessed using the naive forecasting model, predicting the next day using the observations from the same day in the previous week. The MSE score for the three time series are presented in Table 4, showing that the Industry and Household time series are approximately equal in terms of data complexity. The Cabin time series have a higher MSE compared to the latter two. A reasonable result, as the Cabin time series has less of a clear repeating pattern, and contains a increased number of outliers, seen in Figure 24.

Table 4: MSE of the naive predictions of the normalized full dataset, for all three end-users

Time series	Mean squared error
Industry	0.00926
Household	0.00891
Cabin	0.01381

The naive predictions of observations for all time series in the Elvia dataset for a period of two weeks are shown in the Fig. 28. For the Industry time series, the naive predictions performs with reasonable accuracy, since the time series have a smooth and regular weekly pattern, reflected in the MSE score. The plot of the naive predictions for the Cabin time series show a rather poor fit, as confirmed by the MSE scores reported in Tab. 4, indicating increased complexity. This can be expected, as the characteristics of the Cabin time series differ from the two other end-users, both in terms of customer activity and strength of the daily and weekly seasonality.

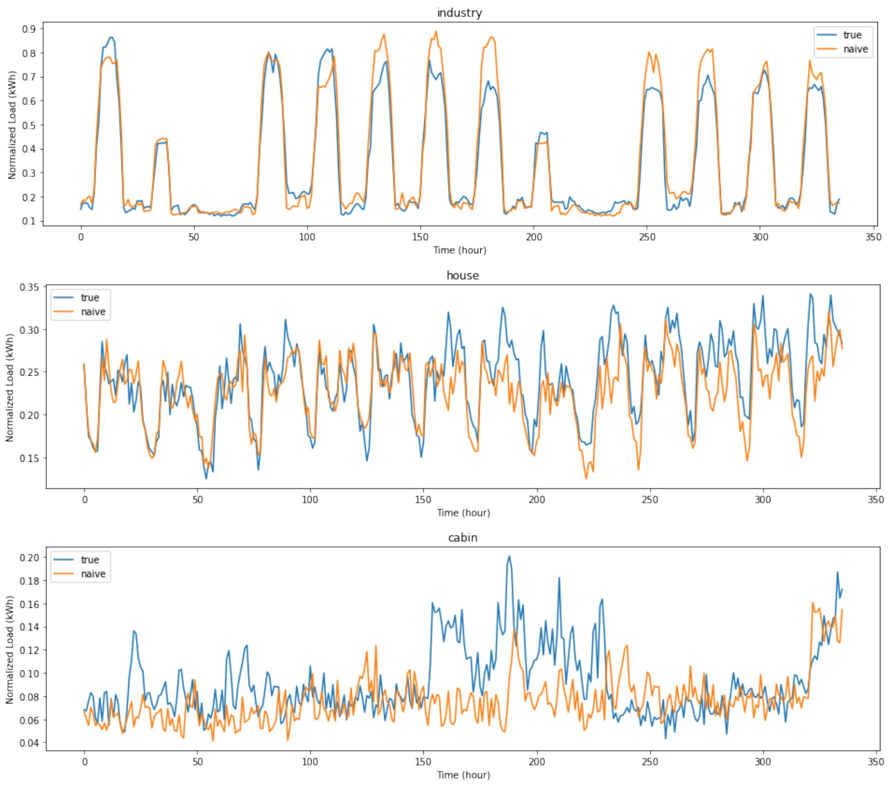


Figure 28: Naive forecasts for a period of two weeks for the three different end-user time series in the Elvia dataset. The blue line corresponds to the true observations, and the orange line is the naive predictions.

14 Models

Each model described in this section is designed to construct multi-step probabilistic forecasts in the form of 90% prediction intervals, i.e. using $\alpha = 0.1$. Specifically, prediction intervals for the next 24 hours of each day in the test datasets are constructed, based on the previous seven days of load measurements. For the multivariate dataset, temperature forecasts of these days are also included as exogenous variables.

In the two following subsections, the model parameter configurations are presented and discussed. First, the configurations of the neural network-based models are presented collectively. Subsequently, the implementation and configuration of the SARIMA models are presented.

14.1 Neural Network-based Models

The neural network-based models are implemented using Keras (Chollet, 2015), a high-level API of TensorFlow 2, which is a Python-based open source deep learning library. The architecture of the TCN and LSTM networks is kept fixed for all models, and the optimal network hyperparameters, such as learning rate, batch size, and layer units, are found by a random hyperparameter search, randomly selecting different parameter configurations from specified intervals and choosing the configuration that achieves the highest prediction interval modeling performance on the validation dataset. For the models construing prediction intervals using pinball loss-guided neural networks, i.e. the EnCQR and QRNN models, the prediction interval coverage and width are evaluated when choosing the optimal hyperparameter configuration (for the EnCQR models, the conformalized intervals are used). For the EnbPI models, the network configuration producing the lowest MSE is chosen. Table 5 reports the chosen configurations for the neural network-based models.

Loss Function

All neural networks are trained using the Adam optimizer. Two different loss function are used, depending on whether the network is to produce point- or probabilistic forecasts. The neural networks producing probabilistic forecasts using quantile regression are designed to optimize the pinball loss defined in Eq. (21), where any desired quantile level can be estimated by altering the q parameter. Prediction intervals are constructed by estimating two quantile levels using the pinball loss-guided neural network, where the corresponding confidence level is the difference between the two quantile levels. To reduce the computational complexity, the neural networks are constructed to directly predict several quantile levels, instead of fitting individual networks for each quantile. This is done by modifying the pinball loss, averaging the loss over all test observations and all target quantiles. The resulting neural networks only need to be trained once to obtain both target quantiles. The neural networks in the QRNN and EnCQR models are both implemented using the total average pinball loss function, defined in Eq. (23).

The underlying neural networks in the EnbPI model produce point forecasts, and are therefore trained to optimize the MSE loss, defined in Eq. (12). The MSE is a popularly used loss function in deep learning models for time series forecasting, which predict the mean expected value (Gasparin et al., 2019).

Table 5: Optimal model configurations for neural network-based models for both datasets. The acronyms in the table are: N_h: number of units in LSTM layers, η: learning rate, λ_2: L2 regularization parameter, b: batch size, and N_f: number of filters in convolutional layers in each residual block.

		Portugal dataset							
TS	**Model**	**Network configuration**							
		LSTM				TCN			
		N_h	η	λ_2	b	N_f	η	λ_2	b
Stat. 250	EnCQR	118	$7.2e^{-4}$	$1.0e^{-5}$	24	1	$5.47e^{-3}$	$1.0e^{-4}$	24
	QRNN	90	$7.5e^{-4}$	$5.6e^{-4}$	32	4	$2.34e^{-3}$	$5.7e^{-5}$	56
	EnbPI	100	$8.0e^{-4}$	0.1	8	1	$1.8e^{-3}$	$4.3e^{-3}$	24
Stat. 77	EnCQR	179	$9.92e^{-4}$	$2.85e^{-5}$	12	2	$2.1e^{-3}$	$6.7e^{-4}$	24
	QRNN	124	$6.0e^{-4}$	$4.9e^{-3}$	32	6	$1.2e^{-3}$	$7.2e^{-4}$	64
	EnbPI	96	0.005	0.001	24	1	$8.32e^{-4}$	$4.3e^{-3}$	16
Stat. 50	EnCQR	92	$2.8e^{-3}$	$4.32e^{-5}$	24	1	$2.1e^{-3}$	$3.58e^{-3}$	16
	QRNN	96	$5.7e^{-4}0$	$7.1e^{-4}1$	32	5	$2.4e^{-3}$	$6.0e^{-4}$	56
	EnbPI	89	$1.96e^{-4}$	$3.5e^{-4}$	16	2	$5.4e^{-4}$	$5.0e^{-5}$	16
Stat. 90	EnCQR	96	$1.1e^{-4}$	$1.77e^{-4}$	24	9	$2.8e^{-3}$	$6.0e^{-5}$	12
	QRNN	112	$8.1e^{-4}$	$9.2e^{-5}$	64	7	0.001	$7.0e^{-4}$	64
	EnbPI	115	$2.9e^{-4}$	0.077	16	1	$3.47e^{-3}$	$4.7e^{-3}$	12
Stat. 27	EnCQR	118	0.003	$5.7e^{-4}$	24	1	0.001	$6.8e^{-5}$	12
	QRNN	200	$6.0e^{-4}$	$2.1e^{-4}$	32	8	$5.0e^{-4}$	$1.2e^{-4}$	32
	EnbPI	136	$6.5e^{-4}$	$3.4e^{-4}$	12	3	$4.8e^{-3}$	$1.2e^{-4}$	16
		Elvia dataset							
TS	**Model**	**Network configuration**							
		LSTM				TCN			
		N_h	η	λ_2	b	N_f	η	λ_2	b
Industry	EnCQR	150	$1.65e^{-3}$	$3.789e^{-4}$	12	1	$9.9e^{-4}$	$5.88e^{-4}$	16
	QRNN	80	$8.9e^{-4}$	$4.95e^{-4}$	32	4	$9.4e^{-4}$	$4.3e^{-4}$	64
	EnbPI	20	0.01	$9.9e^{-3}$	24	22	$8.09e^{-3}$	$1.09e^{-5}$	12
Househ.	EnCQR	47	$6.5e^{-4}$	$1.1e^{-5}$	16	9	$9.0e^{-3}$	$5.88e^{-2}$	12
	QRNN	198	$9.0e^{-4}$	$3.49e^{-2}$	32	6	$1.2e^{-3}$	$9.64e^{-2}$	64
	EnbPI	98	$7.9e^{-4}$	$1.0e^{-4}$	12	3	$4.1e^{-3}$	$3.0e^{-4}$	12
Cabin	EnCQR	97	$5.7e^{-4}$	$2.0e^{-4}$	16	5	$8.7e^{-3}$	0.066	16
	QRNN	98	$5.10e^{-5}$	$6.70e^{-4}$	64	4	0.001	$1.0e^{-4}$	32
	EnbPI	72	0.012	$9.1e^{-4}$	12	3	$3.9e^{-3}$	$3.7e^{-3}$	12

Regularization

Regularization terms are frequently included in network loss functions to prevent overfitting and improve the generalization capabilities of the model. For all models, the same L2 penalty regularization term is applied to the input, hidden and output weights. For all networks, the λ_2 value (the magnitude of the term that penalizes the L2 norm of the weights) is determined during hyperparameter tuning, where the value for λ_2 is randomly chosen from the interval $[0, 0.1]$ using a logarithmic scale.

Additionally, batch normalization regularization is performed after each convolutional layer in the TCN residual blocks.

Initialization

All network weights and biases are initialized using He's uniform variance scaling initializer (He, Zhang, Ren, & Sun, 2015) and zero initialization, respectively.

Ensemble Learners

The EnbPI and EnCQR models consist of an homogeneous ensemble of neural networks, where the number of ensemble learners used in the models is fixed and determined based on the size of the training dataset. (Xu & Xie, 2020) state that producing short and stable prediction intervals requires a large number of ensemble learners and suggest $B = 20-30$. For both datasets, the training data consists of one year of hourly load measurements. To ensure that the underlying neural networks are trained on a sufficient amount of data, a smaller number of member learners is chosen, $B = 12$, resulting in the networks being trained on approximately one month of data, which is thought to be a logical and appropriate setup.

Diversity between the ensemble learners is obtained by training them on different subsets of the training datasets. In the original theory on the EnbPI algorithm presented by (Xu & Xie, 2020), subsets are constructed by sampling with replacement. However, sampling with replacement is not possible when using TCNs and LSTM networks as underlying regression algorithms, since these networks require the input samples to be presented sequentially to learn the temporal structures within the time series. Therefore, both the EnbPI and EnCQR models are presented with subsets created as described in lines 1-4 in Algorithm 4.

To obtain independent subsets, i.e. subsets where no time steps are contained in more than one of the subsets, the training data is first divided into 12 equal parts before converting the time series into input-output sample sequence pairs, as required in supervised learning problems. Consequently, the first week of each subset cannot be predicted since historical observations are unavailable. In total, 12 weeks of the original training dataset cannot be predicted, and the number of out-of-sample residuals will therefore be reduced to $T - (12 \times 168)$, where T is the length of the training data. A simple illustration of how subsets are created is shown in Fig. 29.

All ensemble models use the mean aggregation function, and throughout the experiments, the batch size parameter is fixed at $s = 1$, resulting in the residuals being updated after every day in the test dataset is predicted.

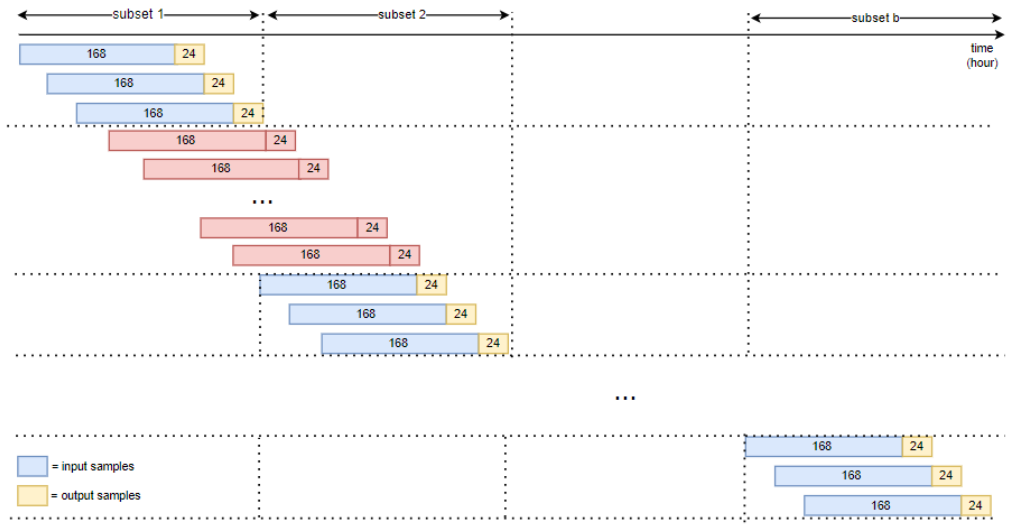


Figure 29: A simple illustration on how independent ensemble subsets are sampled, where each subset contains the same amount of input-output sample pairs. The red sample pairs corresponds to the time steps contained within two subsets, which are discarded.

Convolutional hyperparameters

The input samples in both datasets include 168 observations and the TCN's receptive field should therefore not exceed the length of the input observations. The kernel size and dilation factors are therefore set to $k = 2$ and $d = [1, 2, 4, 8, 16, 20, 32]$, respectively, resulting in the use of seven residual blocks in the TCNs. The number of filters in the convolution layers is determined via hyperparameter tuning and set equal for all residual blocks.

LSTM

The Keras library supports both stateful and stateless LSTM networks. In the Keras LSTM layer documentation, `https://keras.io/api/layers/recurrent_layers/lstm/`, it is specified that in a stateful LSTM layer, the last state for each sample at index i in a batch will be used as initial state for the sample of index i in the following batch. For a stateless LSTM layer, the initiate hidden states are reset between each batch and, therefore, the network cannot learn time dependencies spanning across different batches.

The LSTM networks implemented in the experiments are chosen to be stateless. The datasets used in the experiments are transformed into input and output samples with size of 168 and 24, respectively, and using stateless LSTM layers results in the network only constructing prediction based on the last 168 observations, whereas in a stateful LSTM, the network can look further into past values. Using stateless LSTM layers does not leverage the full potential of the LSTM network, but makes the comparison between the performance of the TCN and LSTM network more direct.

Quantile levels

All QNN models are constructed to estimate the 0.05 and 0.95 quantile functions, representing the lower and upper bands of the 90% prediction intervals.

The quantile levels of the underlying quantile networks in the EnCQR algorithm can be tuned as additional network hyperparameters to construct shorter prediction intervals, as described in Section 10.1. The resulting quantile levels, which are used to estimate the quantile regression prediction intervals in the EnCQR models, are presented in Table 6. Interestingly, the EnCQR models tend to choose quantile levels below the target coverage level of 90%, indicating the the underlying neural network ensemble estimators tend to be too conservative, resulting in unnecessarily wide prediction intervals. The experimental results, presented in Tab. 8 in Section 16, show that the EnCQR models obtain approximately valid coverage for all time series, demonstrating that these models significantly improves the designed coverage level of the underlying neural networks specified in Tab. 6.

Table 6: Upper and lower quantile levels estimated by the underlying quantile neural networks in the EnCQR models. The corresponding confidence level for the interval constructed by the lower and upper quantile functions is reported under coverage (%).

Portugal dataset						
	LSTM			TCN		
	$\hat{q}_{lo}$	$\hat{q}_{hi}$	coverage	$\hat{q}_{lo}$	$\hat{q}_{hi}$	coverage
Station 250	0.100	0.900	80.0	0.050	0.950	90.0
Station 77	0.145	0.815	67.0	0.060	0.920	86.9
Station 50	0.091	0.931	84.0	0.200	0.920	72.0
Station 90	0.210	0.980	77.0	0.050	0.940	89.0
Station 27	0.190	0.990	80.0	0.300	0.999	69.9
Elvia dataset						
	LSTM			TCN		
	$\hat{q}_{lo}$	$\hat{q}_{hi}$	coverage	$\hat{q}_{lo}$	$\hat{q}_{hi}$	coverage
Industry	0.100	0.887	78.7	0.080	0.940	86.0
Household	0.250	0.900	65.0	0.0411	0.995	95.4
Cabin	0.152	0.915	76.3	0.075	0.995	92.0

Network training

After the network hyperparameter configurations are selected, the networks are trained for 10 runs using random and independent network parameter initializations, evaluating the results on the test data after each run. For both metrics described in Section 15, the mean and standard deviation over all runs are reported, as well as the best results[15] obtained. The experiments are repeated for ten independent runs because of the stochastic nature of the networks; The networks weights are randomly initialized, resulting in a different performance for the individual runs using the same model configuration and training data. By reporting the average of several runs, a more solid assessment of the model performance is obtained.

[15]The run with the best result is the one with a prediction interval coverage equal or closest to its designed level, with the smallest average interval length.

During training, early stopping is used for all models, terminating the training process if there is no improvement of the monitored validation loss for 10 epochs. In the experiments, the model performance is evaluated at each epoch using the validation dataset. When the training is complete, the network parameters from the epoch that obtained the best performance on the validation dataset are chosen as the optimal network parameters, and are used to predict the observations in the test dataset.

Table 7: Seasonal ARIMA model parameters, (p,d,q)x(P,D,Q)m. For the Elvia dataset, historical temperature forecasts are added as exogenous variables, the extended model is termed SARIMAX.

Portugal dataset		**Elvia dataset**	
Time series	Model	Time series	Model
Station 250	SARIMA(3,1,1)x(1,1,1)24	Industry	SARIMAX(1,0,2)x(1,1,1)24
Station 77	SARIMA(2,0,2)x(1,1,1)24	Household	SARIMAX(4,1,1)x(1,1,1)24
Station 50	SARIMA(2,1,4)x(1,1,1)24	Cabin	SARIMAX(2,1,1)x(1,1,2)24
Station 90	SARIMA(4,0,0)x(1,1,1)24		
Station 27	SARIMA(3,1,2)x(1,1,1)24		

14.2 SARIMA Models

The seasonal ARIMA models are implemented using the SARIMAX function from the *statsmodels* module in Python. The SARIMAX function fits a model using the provided training data, and predicts a specified number of out-of-sample forecasts from the end of the training samples index. For the multivariate dataset, both historical load and historical records of the exogenous variables are presented to the model. The model orders for each time series are determined by analysing the ACF and PACF plots, together with the AIC criterion, choosing the models that obtain the lowest AIC score. The model orders for all time series in both datasets are reported in Table 7.

As discussed in Section 6.5.1.4, the width of the prediction intervals obtained from autoregressive models tends to increase for each step in the forecasting horizon due to the lack of validity of the assumptions made about the forecasting distribution. The predictions constructed by autoregressive models are based on estimates of the standard deviation using the model residuals, which for increasing forecasting horizons tend to become less accurate. Autoregressive models construct forecasts based on a number of previous values and residuals of the time series, determined by the model orders. For multi-step forecasting, the forecasts are added to the model's available history as true observations, and the following predictions are therefore based on previous predictions, often resulting in an error accumulation, contributing to the widening of the prediction intervals (illustrated in the top plot in Fig. 30).

This thesis places its main interest in multi-step forecasting, predicting the next 24 hours for each day in the test dataset. All time series from both datasets used in the experiments are divided into train, validation, and test dataset, where the validation sets are used to find the optimal hyperparameters of the neural network-based models and to save the best model during training. Contrarily, the ARIMA models do not use cross-validation procedures for model selection, as they adopt the ACF and PACF plots and AIC criterion. To obtain identical prediction setups for the neural network-based models and the SARIMA models, the latter is fitted only using the training dataset, but due to the autoregressive nature of these models, and the fact that they are not presented with samples in the form of input-output pairs, the observations in the validation set need to be made available for the models. To achieve this, the statsmodels *append* function is utilized. The append function stores the results for all training observations and extends the model's available historic observations without refitting the model parameters.

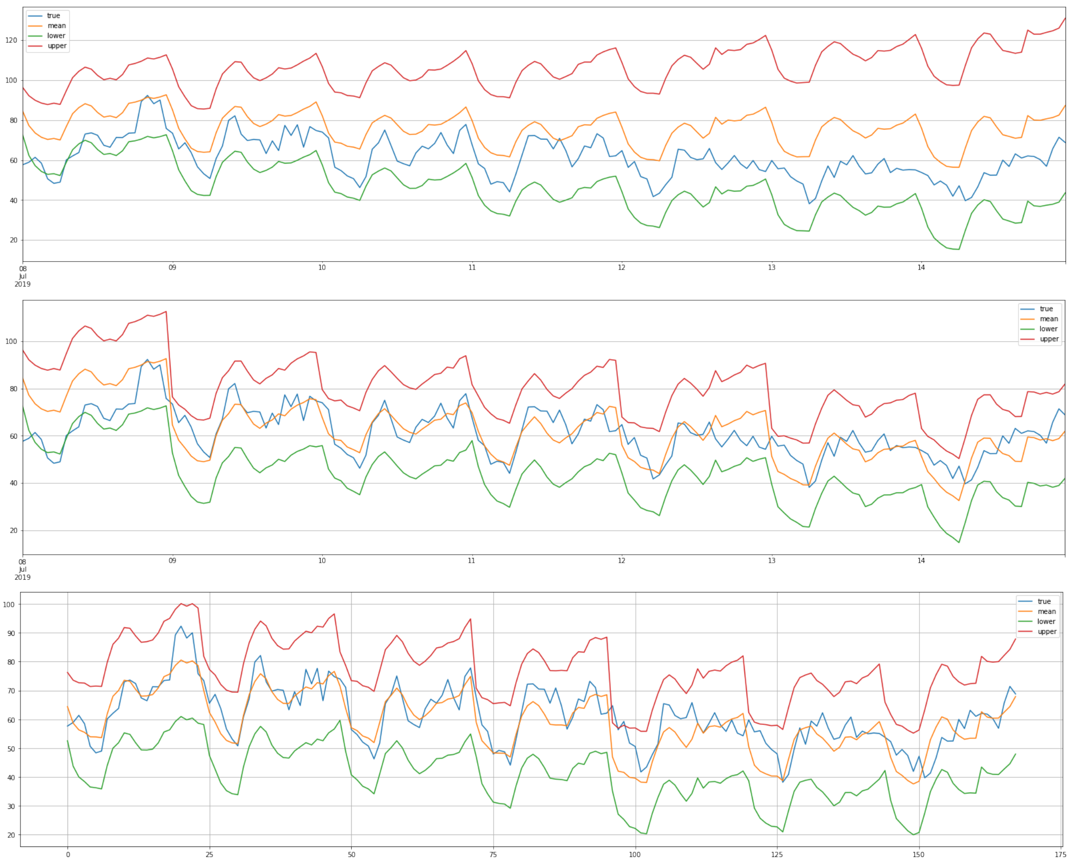


Figure 30: Illustrations of a SARIMA model producing wide and shifted prediction intervals. The plots show three models fitted using the Household time series from the Elvia dataset, with training data from 1 June 2018 to 31 May 2019. All models predict test observations from 8-14 July 2019. The model in the top plot is not presented with new data after being fitted, and predicts all test observations at once. The model in the middle plot predicts 24 hours ahead for each day in the test data, and is presented with the actual observations after the predictions are made. However, the observations from 1-31 June 2019 are not presented to the model and, thus, it performs poorly in the beginning. In the bottom plot, the model is presented with observations from 1 June to 31 June 2019 after being fitted, as well as the actual observations after the predictions are made. The prediction interval is no longer shifted as before, since the model is presented with all required data. The blue line corresponds to the actual observations, the yellow line is the predicted mean, and the red and blue lines represent the upper and lower prediction interval bounds, respectively.

Furthermore, ideally, one SARIMA model should be fitted for each day in the test data to obtain optimal results. However, this is infeasible due to the size of the test dataset. During the experiments, the results showed that if all values in the test sets were predicted using a single SARIMA model only presented with the original training data, the results were uninteresting, since the prediction intervals quickly became very wide and uninformative, reaching 100% coverage almost immediately. Additionally, the predictions of the first time steps in the test data were often shifted, not matching the true observations, as illustrated in the middle plot of Fig. 30. From the figure, one can see that the model was able to recover, matching the true observations after a few time steps. However, not presenting the models with the required data does not create a fair basis for comparison.

To avoid the above-mentioned problems, a single SARIMA model is fitted using the training dataset, and as the days in the test dataset are predicted, the previous actual observations for the test days and the validation data are made available for the model using the append function. The use of the append function allows for only one model to be constructed for each time series, while avoiding the widening of the prediction intervals. Additionally, as discussed in the data preprocessing sections, the first observations corresponding to the sample input size of the test data cannot be predicted by the neural networks because historical data is unavailable. Therefore, the days with missing historical data are not predicted by the SARIMA models, but added to the model's available observation using the append function. The bottom plot in Fig. 30 shows the prediction intervals obtained using the append function, providing the model with both the validation data and the actual test observation after each 24 hours are predicted.

Unlike the neural networks, SARIMA models are deterministic and are only fitted once for each time series. Therefore, the experimental results from a single run of the SARIMA models for each time series are reported.

15 Evaluation Metrics

When constructing probabilistic forecasts using several approaches, it is critical to properly assess the predictive ability of the individual forecaster in order to rank the competing forecasting approaches (Gneiting et al., 2007). Probabilistic forecasts in the form of prediction intervals can be evaluated by assessing the two complementary aspects of the intervals: coverage and width. In some cases, the forecaster might only be interested in the coverage of prediction intervals, and the predictions can be evaluated only in terms of coverage. However, to properly assess the quality of a prediction interval, both the coverage and width should be considered jointly.

As described in Section 6.4, the ideal prediction interval must maximize the sharpness of the predictive distributions, subject to calibration (Gneiting & Katzfuss, 2014), where the sharpness of prediction intervals refers to how tightly the interval covers the actual distribution. A prediction interval is calibrated if the interval coverage of a new test point is guaranteed to be greater or equal to the designed confidence level. Thus, probabilistic forecasts in the form of intervals should be as narrow as possible while reflecting the designed coverage level.

For a quantitative evaluation of the performance of the models described in Section 14, two frequently used probabilistic forecasting performance measures are chosen (Shepero, Van Der Meer, Munkhammar, & Widén, 2018):

- **Prediction interval coverage probability** (PICP): The prediction interval coverage of a single data point level is binary: the data point either lies within the bounds of the prediction interval, or not. Therefore, to assess the prediction interval coverage of several data points, e.g. a time series, the coverage of the individual points in the sequence must be averaged over the whole set of point. The *PICP* score can be used to evaluate prediction interval coverage, and is calculated as the sum of all observations lying within the bounds of the prediction interval, divided by the total number of observations in the dataset, n_t.

$$\text{PICP} = \frac{1}{n_t}\sum_{i=1}^{n_t} c_i, \quad c_i = \begin{cases} 1, & y_t \in [L_i, U_i] \\ 0, & y_t \notin [L_i, U_i] \end{cases} \tag{32}$$

The PICP score is a value between zero and one, where a prediction interval that covers all observations in the dataset has PICP score equal to one.

- **Prediction interval normalized average width** (PINAW): The average width of a prediction interval is defined as the difference between the upper (U) and lower (L) interval limits, divided by the total number of observations in the dataset:

  $$\text{PIAW} = \frac{1}{n_t}\sum_{i=1}^{n_t}(U_i - L_i) \tag{33}$$

  The unit and numerical value of the PIAW is dependent on the unit and scale of the variable being predicted. To compare between intervals, the prediction interval average width must be normalized:

  $$\text{PINAW} = \frac{1}{n_t R}\sum_{i=1}^{n_t}(U_i - L_i) \tag{34}$$

  where $R = y_{max} - y_{min}$ is the difference between the maximum and the minimum actual value of the predicted variable.

The use of the PICP alone does not provide a good performance measure, since very wide intervals can have high coverage, but are less informative. The PICP together with the PINAW can therefore be used as an assessment of the prediction interval quality, where the best forecasting model minimizes the PINAW, while simultaneously maximizing PICP.

In this thesis, we focus on constructing valid prediction intervals for electricity load time series. Therefore, when evaluating the predictive abilities of the different models described in Section 14, we first compare the prediction interval coverage, then prediction interval width. The best-performing model is the model producing prediction intervals with coverage equal or closest to its designed level, with the smallest average interval length.

16 Experimental Results

This section presents and discusses the results obtained from the experiments described above. First, the results for the individual datasets are analysed in Section 16.1. Section 16.2 discuss more specific aspects of the experimental results, and examines the effect of the conformalization step in the EnCQR algorithm. The quantitative results of the experiments are presented in Table 8.

Table 8: Experimental results for both datasets. For the neural network-based models, the reported results are based on 10 runs, whereas the results from the SARIMA models are reported from a single run, due to their deterministic nature. The best results are highlighted in bold, referring to the prediction interval with coverage equal or closest to its designed level, with the smallest average interval length.

Portugal dataset

TS	Model	PICP / PINAW				
		LSTM		TCN		SARIMA
		Best run	Mean (std.)	Best run	Mean (std.)	Result
Station 250	EnCQR	**0.910 / 0.240**	0.900 (0.039) / 0.264 (0.023)	0.911 / 0.355	0.879 (0.018) / 0.327 (0.022)	-
	QRNN	0.842 / 0.175	0.776 (0.059) / 0.167 (0.0375)	0.886 / 0.325	0.793 (0.064) / 0.241 (0.054)	-
	EnbPI	0.901 / 0.241	0.885 (0.009) / 0.220 (0.009)	0.929 / 0.318	0.916 (0.026) / 0.352 (0.044)	-
	SARIMA	-	-	-	-	0.789 / 0.154
Station 77	EnCQR	**0.926 / 0.262**	0.902 (0.029) / 0.259 (0.021)	0.902 / 0.342	0.900 (0.028) / 0.355 (0.003)	-
	QRNN	0.867 / 0.175	0.721 (0.072) / 0.167 (0.050)	0.766 / 0.231	0.685 (0.077) / 0.262 (0.073)	-
	EnbPI	0.935 / 0.287	0.950 (0.009) / 0.318 (0.002)	0.922 / 0.361	0.875 (0.029) / 0.363 (0.030)	-
	SARIMA	-	-	-	-	0.726 / 0.179
Station 50	EnCQR	0.909 / 0.317	0.863 (0.025) / 0.286 (0.020)	0.901 / 0.373	0.887 (0.018) / 0.365 (0.027)	-
	QRNN	0.891 / 0.330	0.858 (0.019) / 0.243 (0.049)	0.848 / 0.256	0.832 (0.011) / 0.261 (0.021)	-
	EnbPI	**0.904 / 0.306**	0.907 (0.008) / 0.318 (0.013)	0.880 / 0.416	0.863 (0.009) / 0.395 (0.034)	-
	SARIMA	-	-	-	-	0.802 / 0.219
Station 90	EnCQR	**0.904 / 0.283**	0.917 (0.042) / 0.332 (0.037)	0.900/ 0.300	0.885 (0.017) / 0.319 (0.016)	-
	QRNN	0.831 / 0.235	0.739 (0.054) / 0.207 (0.038)	0.829 / 0.272	0.775 (0.033) / 0.270 (0.016)	-
	EnbPI	0.910 / 0.293	0.931 (0.009) / 0.308 (0.008)	0.934 / 0.432	0.860 (0.025) / 0.340 (0.016)	-
	SARIMA	-	-	-	-	0.893 / 0.215
Station 27	EnCQR	**0.900 / 0.271**	0.916 (0.014) / 0.303 (0.019)	0.910 / 0.309	0.931 (0.019) / 0.350 (0.037)	-
	QRNN	0.898 / 0.268	0.884 (0.007) / 0.191 (0.031)	0.881 / 0.305	0.868 (0.008) / 0.287 (0.026)	-
	EnbPI	0.933 / 0.289	0.911 (0.009) / 0.265 (0.009)	0.880 / 0.273	0.861 (0.009) / 0.297 (0.015)	-
	SARIMA	-	-	-	-	0.590 / 0.196

Elvia dataset

TS	Model	PICP / PINAW				
		LSTM		TCN		SARIMA
		Best run	Mean (std.)	Best run	Mean (std.)	Result
Industry	EnCQR	**0.954 / 0.464**	0.959 (0.015) / 0.499 (0.022)	0.900 / 0.492	0.902 (0.014) / 0.513 (0.013)	-
	QRNN	0.846 / 0.152	0.828 (0.012) / 0.231 (0.100)	0.803 / 0.347	0.800 (0.007) / 0.396 (0.025)	-
	EnbPI	0.903 / 0.500	0.913 (0.011) / 0.543 (0.053)	0.977 / 0.497	0.972 (0.008) / 0.533 (0.045)	-
	SARIMA	-	-	-	-	0.887 / 0.430
Household	EnCQR	**0.900 / 0.503**	0.871 (0.023) / 0.488 (0.055)	0.957 / 0.753	0.925 (0.031) / 0.863 (0.078)	-
	QRNN	0.803 / 0.146	0.791 (0.024) / 0.251 (0.162)	0.941 / 0.789	0.588 (0.127) / 0.423 (0.126)	-
	EnbPI	0.997 / 0.630	0.999 (0.001) / 0.718 (0.070)	0.900 / 0.710	0.871 (0.027) / 0.701 (0.043)	-
	SARIMA	-	-	-	-	0.894 / 0.148
Cabin	EnCQR	0.913 / 0.434	0.920 (0.031) / 0.490 (0.029)	0.907 / 0.665	0.893 (0.050) / 0.638 (0.085)	-
	QRNN	0.880 / 0.219	0.856 (0.079) / 0.380 (0.157)	0.900 / 0.573	0.736 (0.131) / 0.455 (0.136)	-
	EnbPI	0.984 / 0.621	0.990 (0.005) / 0.694 (0.062)	0.912 / 0.628	0.921 (0.20) / 0.687 (0.087)	-
	SARIMA	-	-	-	-	**0.930 / 0.257**

16.1 Performance on Individual Datasets

16.1.1 Portugal Dataset

The results in Table 8 show that the EnCQR-LSTM models generally have the highest performance for all time series in the Portugal dataset, with the exception of Station 50. The boxplots in Fig. 31 graphically illustrate the spread in prediction interval length and coverage for the results, showing that the coverage boxes for both conformal prediction-based models are centered approximately on the specified coverage level, having smaller variance than the QRNN models for almost all time series. The QRNN models produce the narrowest prediction intervals, but their coverage varies significantly between runs and they significantly undercover for some time series.

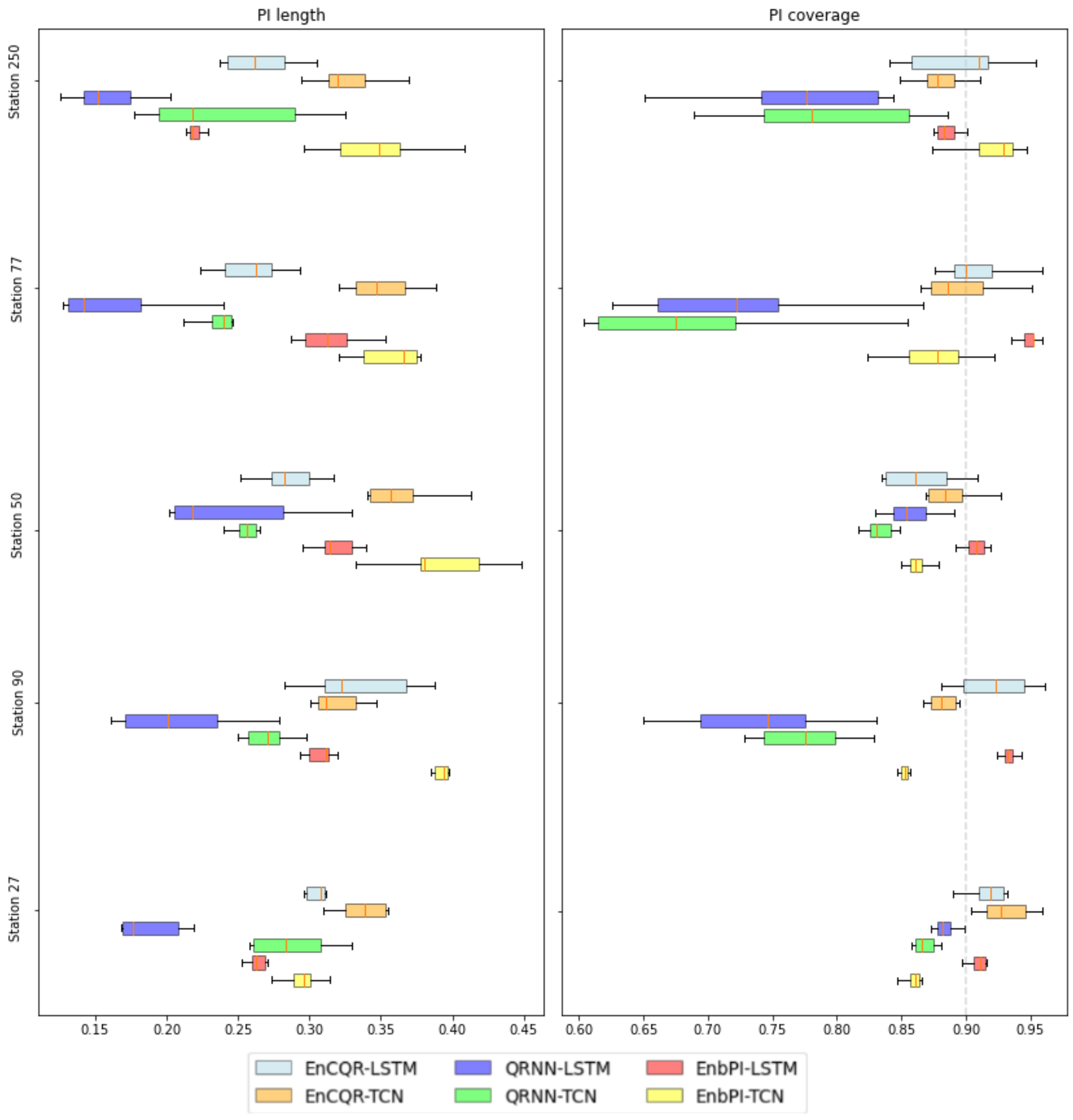


Figure 31: Portugal dataset: Boxplots of average PI length (left) and PI coverage (right) for the neural network-based models on different time series. Each box represents results from ten individual runs. The gray line at 0.9 in the PI coverage plot denotes the target coverage level.

Similar to the QRNN models, the results show that the SARIMA models tend to produce narrow intervals, indicating high confidence, but where the actual coverage is significantly below the designed coverage level for most of the time series. However, for Station 90, the SARIMA model achieves near valid marginal coverage, producing the narrowest intervals of all models.

To further investigate the performance of the neural network models, Fig. 32 depicts a scatter plot showing the relationship between prediction interval coverage and prediction interval length for all time series. In the scatter plot, points in the bottom right indicate models with the best performance, points in the top left models with the worst performance. Interestingly, the scatter plot shows some clear groupings, indicating the difference in performance between the different methods and between the two different network types: The vertical dashed line represents a relatively clear separation between the conformal prediction-based methods and the QRNNs, showing that conformal prediction-based methods produce intervals with coverage centered around the designed coverage level. The diagonal dotted line separates well the two different network types, where the TCNs are generally shown to produce wider intervals compared to the LSTM networks.

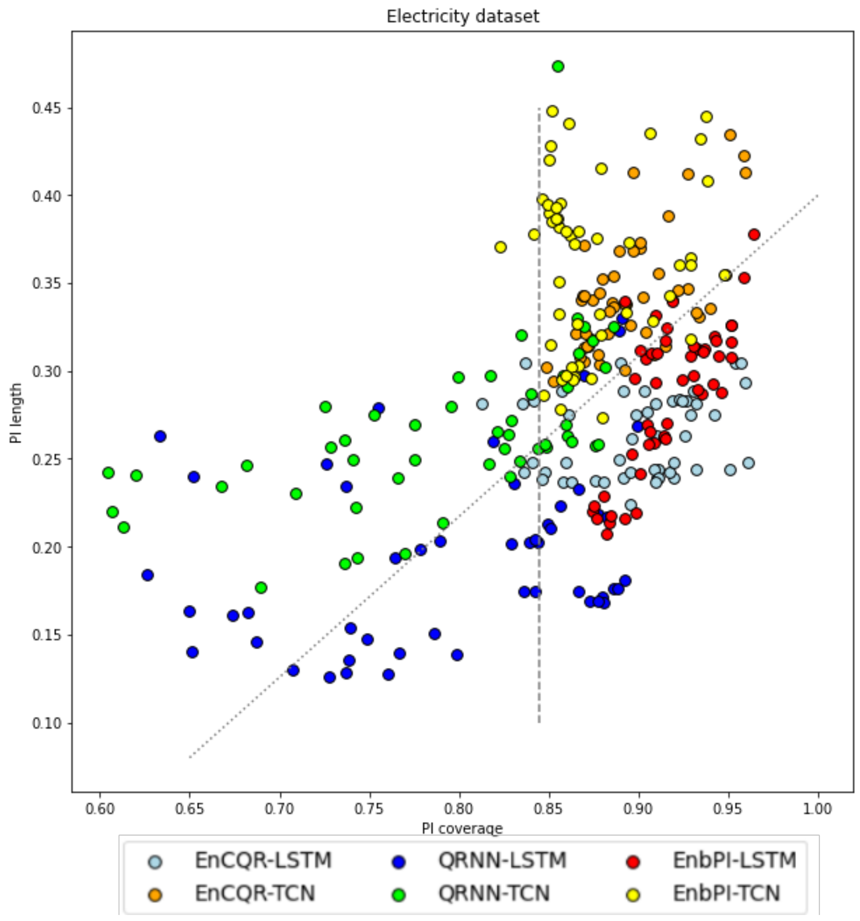


Figure 32: Scatter plot showing the relationship between prediction interval coverage and length for all time series in the Portugal dataset. Each point represents the results obtained by training one of the 10 instances of a neural network model to predict one of the 5 time series

In the scatter plot, almost all points representing the EnCQR-LSTM models are located in the lower right corner of the plot, indicating that these models produce prediction intervals with the highest coverage and narrowest width. Similarly, the EnbPI-LSTM points are primarily located in this region, but the vertical spread is slightly larger for these models compared to the former, displaying large variability in prediction interval width. A small group of the QRNN-LSTM points is located around the target coverage, showing slightly lower prediction interval width than the former two algorithms and demonstrating that the QRNN-LSTM models can construct significantly narrower, approximately valid intervals. However, most of the points for this model are located at lower coverage levels. Additionally, the scatter plot shows a positive and approximately linear association between prediction interval coverage and width, indicating that when the prediction interval coverage increases, so does the prediction interval width. This agrees with statements made in Section 6.4.

16.1.2 Elvia Dataset

Similarly to the Portugal dataset, the overall best performing model for the Elvia dataset is the EnCQR-LSTM, which produces the sharpest valid prediction intervals for two of the three time series. Notably, the difference in prediction interval width between the EnCQR-LSTM and EnbPI-LSTM is for the Elvia datasets larger than for the Portugal dataset, where the two models produced almost identical results. As for the Portugal dataset, the QRNN-LSTM and QRNN-TCN models produce the narrowest prediction intervals for all time series and, also here, the variation in prediction interval coverage is considerably high, as shown in Fig. 33. In terms of validity, the coverage boxes of conformal prediction-based models are in Fig. 33 located at target coverage or higher coverage levels.

From Table 8 one can see that the prediction intervals for the Elvia time series are for the conformal prediction-based models remarkably wider compared to the Portugal time series. Additionally, the actual coverage of the conformal prediction-based models is significantly higher for the Elvia dataset, where the mean coverage of the ten runs for some time series is close to 100%. As previously stated, there is generally a strong relationship between interval coverage and width, and the significant *overcoverage* by the conformal prediction-based models is reflected in the interval widths, giving some explanation of the increased prediction interval width.

For the Cabin time series, the best performing model is the SARIMA model, producing the most narrow prediction intervals with coverage above the target level. The overall performance of the SARIMA models is higher for the Elvia dataset than for the Portugal dataset, since coverage above or approximately on target level is obtained for all three time series. Both the SARIMA and the QRNN-LSTM models produce significantly narrower intervals compared to the other models for the Elvia time series.

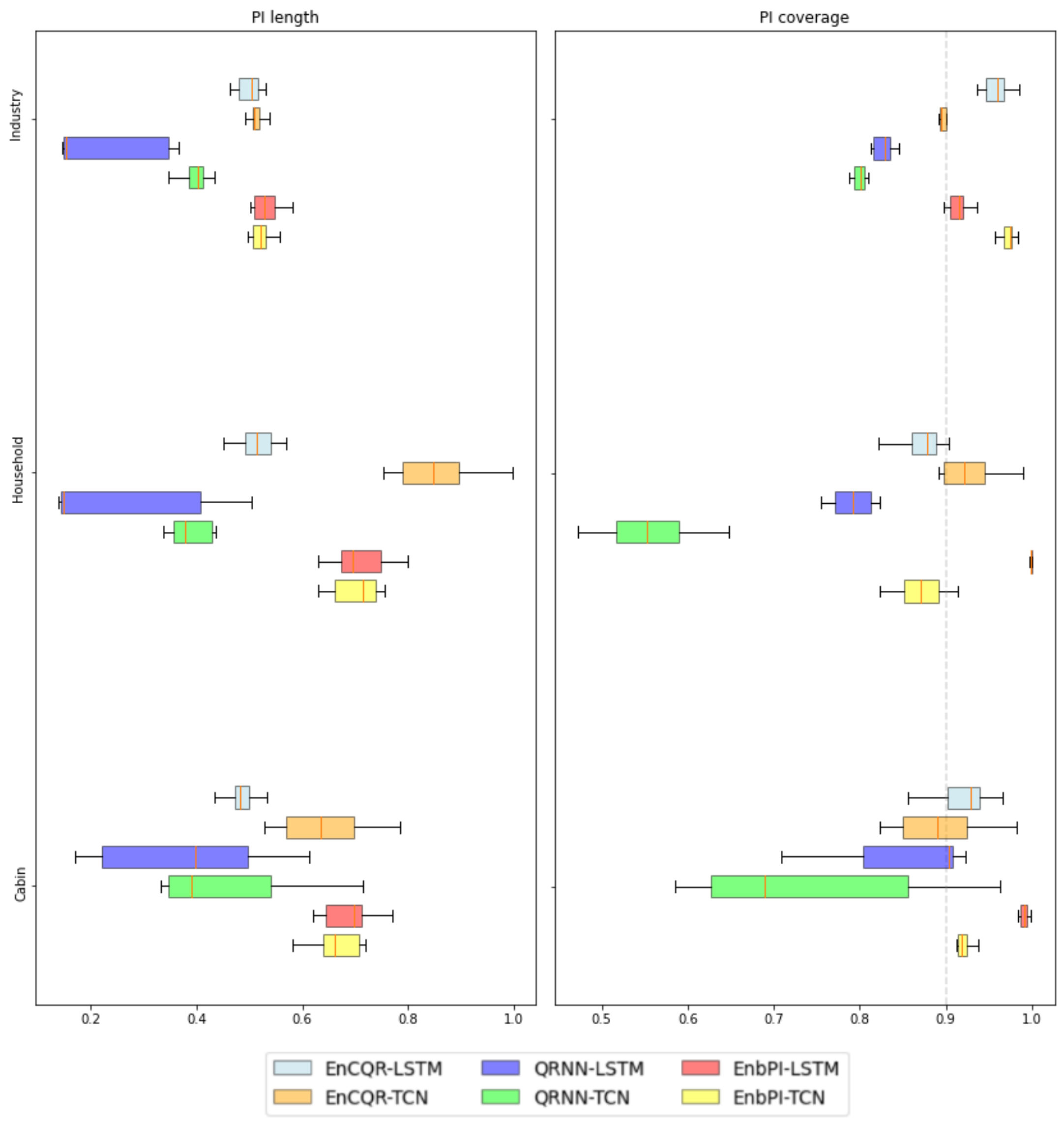


Figure 33: Elvia dataset: Boxplots of average PI length (left) and PI coverage (right) for all time series. Each box contain results from ten individual runs. The gray line at 0.9 in the PI coverage plot show the designed coverage level. The results for the SARIMA models are not shown, since they are deterministic, and only trained once.

Fig. 34 depicts a scatter plot showing the relationship between prediction interval coverage and prediction interval length for all time series in the Elvia dataset. Unlike in the scatter plot for the Portugal time series, now the groupings in Fig. 34 is not as distinct as before: the separation between the EnbPI-LSTM and EnbPI-TCN models are not very apparent anymore, but the LSTM network is still the best performing network type for all methods. The correlation between coverage and length is for the Elvia time series not as strong as for the Portugal dataset, possibly due to the significant increase in prediction interval width for the conformal prediction-based models.

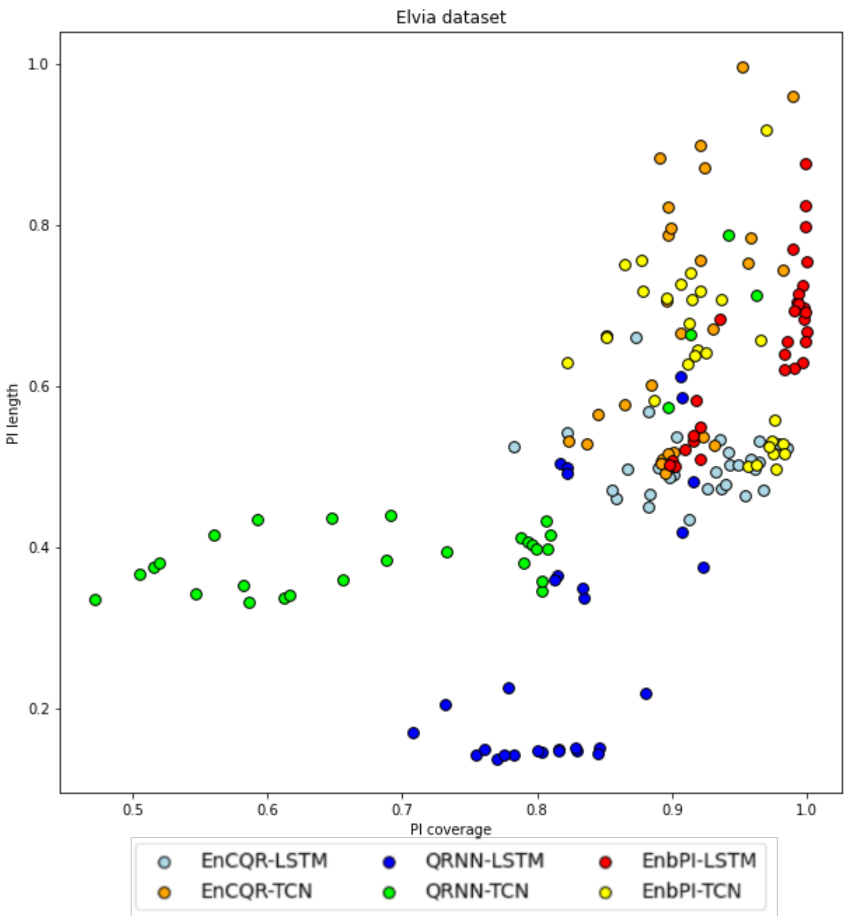


Figure 34: Scatter plot showing the association between prediction interval coverage and length for three end-users in the Elvia dataset. The results for all ten runs of the neural network-based models are plotted.

16.2 Discussion

16.2.1 Prediction Interval Coverage and Width

From Table 8 and Fig. 31-33, it can be seen that the EnCQR models produce the sharpest valid prediction intervals, but the difference in performance between the EnCQR and EnbPI models is for most time series minor. The actual coverage of the EnbPI and EnCQR models is concentrated around the designed target coverage level, having a relative small interquartile variance. Both methods successfully construct approximately valid prediction intervals for all time series, regardless of the ensemble regression algorithm used, demonstrating that their capability holds for any data distribution and possibly any underlying regression algorithm. Furthermore, during model validation, when searching for the optimal network hyperparameter configurations, the results from the hyperparameter search showed that both conformal prediction-based methods attained approximately valid marginal coverage for almost any hyperparameter configuration, once more demonstrating the strength of these models. However, the width of the prediction intervals was shown to vary significantly, meaning that the quality of the prediction intervals can be very different, even though valid coverage is obtained.

Contrarily to the EnbPI and the EnCQR models, the actual coverage of the prediction intervals constructed by the SARIMA and QRNN models greatly varies for the different time series, lacking the robustness of the former two. For the QRNNs, the results from the hyperparameter search showed that both the prediction intervals' validity and sharpness greatly depended on the hyperparameter configuration, unlike for the conformal prediction-based models. The QRNN models are not constrained by the coverage guarantee, as the conformal prediction-based models are, and can therefore construct very narrow intervals, but prediction interval width alone does not indicate the quality of the prediction, because narrow prediction intervals are useless if the actual observations fall far outside their limits more often than specified.

For the Elvia dataset, the coverage of most of the conformal prediction-based models, especially the EnbPI models, is significantly greater than the designed coverage level. The corresponding prediction intervals are wide, signifying that these models can be too conservative. The EnCQR intervals tend to be shorter than the EnbPI intervals but are still unnecessary wide, as the coverage level for some time series is significantly higher than 90%. The most probable reason is that both methods value prediction interval validity more than sharpness by design. The increased prediction interval width for the Elvia time series is further discussed below. Table 6 shows that the EnCQR algorithm tends to choose quantiles with corresponding coverage lower than the target coverage level, consistent with the findings reported by (Romano, Patterson, & Candès, 2019), strengthening the above reasoning.

The results in Table 8 show that the prediction intervals for the Elvia time series are significantly wider than the intervals for the Portugal dataset. A possible explanation might be that the complexity of the time series in the Elvia dataset is increased compared to the Portugal dataset. Section 13.1.2 and 13.2.2 describe the data complexity of the Portugal and Elvia dataset, respectively. A higher MSE of the naive prediction translates into more variation within the time series, indicating increased uncertainty around the individual predictions. Guaranteeing valid marginal coverage is in these situations assumed to be of increased difficulty, and the width of the intervals increases to assure valid coverage. Table 3 and 4 present the MSE scores for the Portugal and Elvia time series, respectively, and show a notable difference between the MSE of the two datasets. The difference in MSE for the Portugal datasets is for the majority of the time series significantly smaller than the MSE for the Elvia time series, indicating that the latter time series are more difficult to predict.

During the examination of the effect of conformalization in the EnCQR algorithm, which is described later in Section 16.2.4, an important discovery was made. For the Elvia dataset, the ensemble prediction intervals (the original intervals, before conformalization) are very narrow and they do not cover most of the true values, as illustrated in Figure 35.

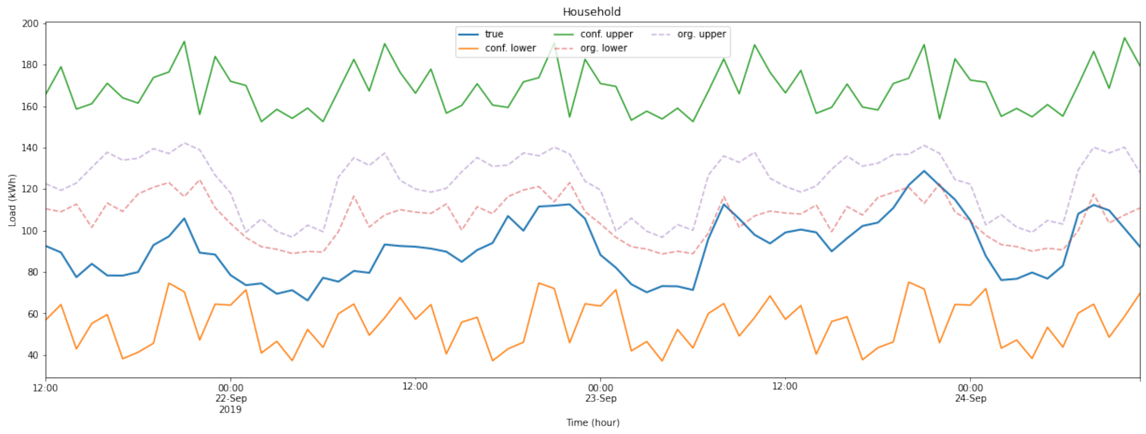


Figure 35: EnCQR-LSTM prediction intervals for the Household time series, showing inflated conformalized intervals.

As described in Section 11.1, the ensemble learners in the EnbPI algorithm are used to estimate the unknown model f, and the assumption placed on them is that they are able to model f with satisfactory accuracy. (Xu & Xie, 2020) report that if this assumption fails, valid coverage can still be obtained if a small batch size parameter is used, but the resulting intervals become inflated if the out-of-sample absolute residuals are large. This also applies to the EnCQR algorithm and from Fig. 35 it is clear that this is the case for the Elvia dataset.

The underlying ensemble quantile neural networks for the Elvia dataset are not able to successfully model the true underlying model, but by using a batch size parameter of $s = 1$, valid coverage is still obtained. This proves that during network validation, when searching for the optimal hyperparameters, careful consideration must be placed on the method used to evaluate the accuracy of the different model configurations. In the experiments, the optimal hyperparameter configurations were found by evaluating the interval coverage and width of the conformalized intervals. However, after a deeper investigation, it is evident that the hyperparameter configurations should be evaluated on the original intervals, before conformalization, to ensure that the performance of the underlying ensemble estimators is satisfactory. Due to time constraints, the hyperparameter search for the Elvia time series could not be re-done, but stress that this should have been done to adequately demonstrate the EnCQR models' actual performance.

Despite the fact that the actual performance could be improved by implementing the hyperparameter search differently, the results demonstrate an important property and the strength of the EnCQR algorithm: Valid coverage can be obtained even if the underlying regression algorithm performs poorly, and if prediction interval coverage is valued more than prediction interval sharpness, this property is beneficial, as valid coverage can be obtained even using simple and possibly inaccurate underlying algorithms. This agrees with the findings by (Xu & Xie, 2020), where they remark the practical usefulness of the EnbPI algorithm, reporting that the empirical results are valid even under potentially misspecified models, as in the case for the Elvia dataset. However, in most real-world situations, probabilistic forecasts are used in high-risk situations, such as electricity production management, where inflated prediction intervals serve no purpose.

When further analyzing the prediction interval modeling performance of the different methods, only the results from the Portugal dataset are considered, as the results on the Elvia datasets do not adequately demonstrate the actual performance of the conformal prediction-based models due to the inflated prediction intervals.

From Fig. 36, which shows the average hourly prediction interval coverage and length variability over the whole test dataset, and all ten individual runs, of all neural network-based models for Station 250, it can be seen that the prediction interval length of the EnCQR and QRNN models correlates with the hourly variability in actual load (illustrated in the top plot of Fig. 36): For the hours when the load is low, i.e. hour 0-5, the prediction intervals are narrower for most models. This is expected, as there is less uncertainty during the hours when the customers' activity is low. Correspondingly, the prediction interval width increases during the central hours of the day, where the spread in actual load is largest, indicating higher uncertainties. The hourly variability in prediction interval width for the EnbPI models, shown in the two bottom right plots in Fig. 36, is not as informative as for the other models: the width of the prediction intervals constructed by these models is constant throughout the day, because the list containing the conformity scores, i.e. the absolute residuals between the predictions and the actual observations, is updated every 24 hours rather than at each hour.

The interpretation of the hourly variability in prediction interval coverage, shown in the plots in the left column Fig. 36, is not as straightforward as for the prediction interval width. For the EnCQR models, the coverage is less during the hours that the prediction intervals are the narrowest, possibly indicating that these models are too confident at these hours.

To further investigate the results, it is possible to look at the plot in Fig. 37, showing the average hourly prediction interval over all hours in the test dataset and all runs, for all models. From Fig. 37, it is clear that the prediction intervals of the EnCQR models are not centered around the actual load of these hours but located at too high load levels, resulting in the actual load observations frequently falling below the lower bound of the prediction interval. The same reasoning can explain the coverage of the QRNN and EnbPI models: the prediction intervals of the EnbPI-TCN models are not centered during the central hours of the day, resulting in poor coverage for these hours. For the QNN-LSTM and QNN-TCN models, some parts of the boxes representing the interquartile range of the actual load are in Fig. 37 located outside the bounds of the prediction interval, explaining the substantial undercoverage seen in Fig. 36 during the central and early hours of the day, respectively.

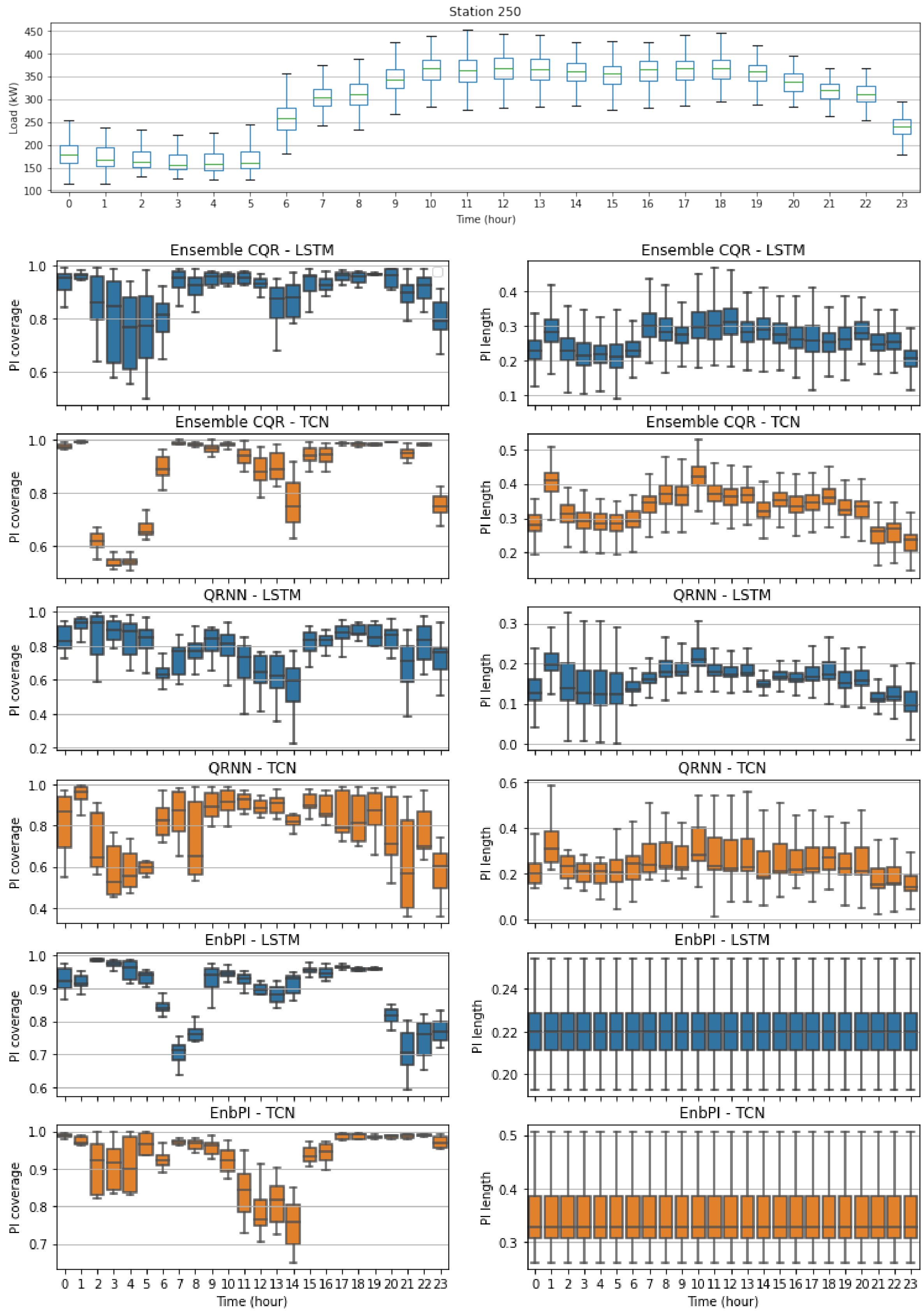


Figure 36: Station 250: Average hourly prediction interval coverage and width variability over all hours in the test dataset, for ten individual runs of all neural network-based models. The top plot show the hourly load variability for all days in the test dataset.

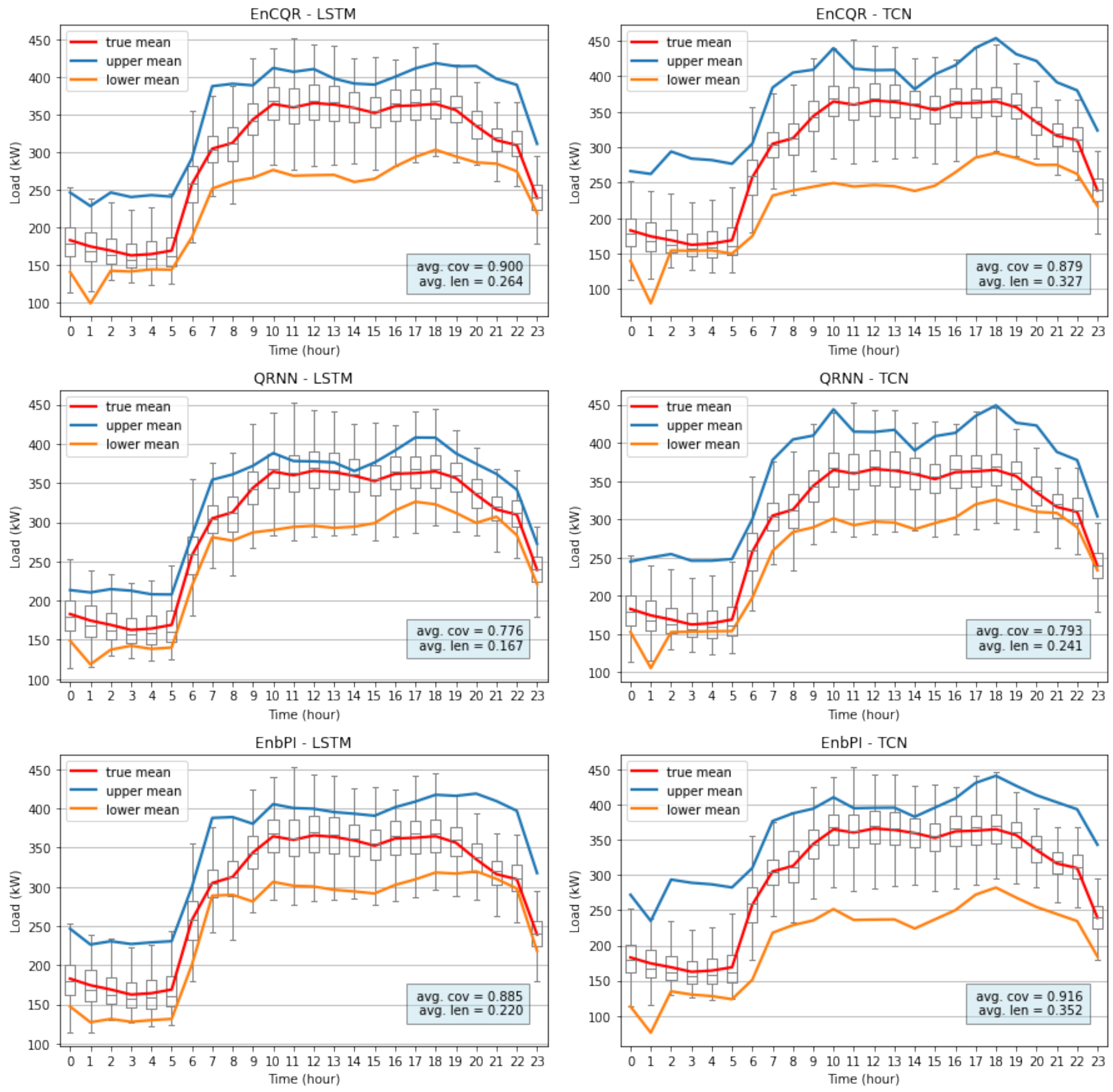


Figure 37: Station 250: Average prediction interval for all hours in the test dataset. The red line represent the mean actual load, and the blue and yellow line represent the average upper and lower prediction interval bounds, respectively, where both lines show the aggregated results from ten individual runs. The underlying boxplot represent the hourly load variation for the test dataset.

16.2.2 Network Type

In general, the results show that LSTM networks outperform the TCNs in terms of prediction interval width, but the coverage of both network types are, on average, approximately equal. For some time series, the difference in prediction interval width varies significantly, i.e. Station 250 (of the Portugal dataset), Household and Cabin (of the Elvia dataset), whereas for Station 27 and Station 90 (of the Portugal dataset) and the Industry time series (of the Elvia dataset), the best results are almost identical.

Overall, the LSTM network tends to outperform TCN, producing higher quality prediction intervals. This result is not very surprising, as it is common that an architecture A performs better than an architecture B on some problems but, on other problems, the situation is inverted, and the average performance of the two architectures will be the same across all possible problem instances, demonstrated by the *no free lunch* theorem (Wolpert, 1996). However, some investigations are performed to find a possible explanation for the difference in performance between the two network types.

One possible reason is that the TCNs might be overfitted, generalizing badly, thus performing worse on the unseen test dataset. To examine if the TCNs are overfitting, the networks are evaluated on the test data and the training data used to fit the networks. If the results for the test data are poor compared to results for the training data, one can conclude that the network is overfitted.

The investigations are performed by looking at the quantile regression-based models for two time series from both datasets. The normalized total pinball loss, defined in Eq. (23), is used to assess the performance of the models, where a lower pinball loss score indicates higher prediction interval quality. The results from the investigations are presented in Table 9, indicating no overfitting for any of the networks.

Table 9: Normalized total pinball loss for quantile levels (0.05 , 0.95). To investigate possible network overfitting, both training and test data are predicted, and the resulting pinball loss scores are used for comparison. The models selected are the ones that produced the best results reported in Table 8.

Portugal dataset					
Time series	**Model**	**Pinball loss**			
		LSTM		TCN	
		TRAIN	TEST	TRAIN	TEST
Station 90	EnCQR	0.0120	0.0092	0.01245	0.0096
	QRNN	0.0062	0.0096	0.0099	0.0104
Station 27	EnCQR	0.0085	0.0102	0.0108	0.0098
	QRNN	0.0076	0.0088	0.0104	0.0105
Elvia dataset					
Time series	**Model**	**Pinball loss**			
		LSTM		TCN	
		TRAIN	TEST	TRAIN	TEST
Industry	EnCQR	0.0149	0.0129	0.0168	0.0156
	QRNN	0.0052	0.0067	0.0154	0.0120
Household	EnCQR	0.0196	0.0145	0.0239	0.0197
	QRNN	0.0048	0.0071	0.0243	0.0215

The EnbPI algorithm avoids overfitting by using ensemble predictors to construct leave-one-out estimations, aggregating together the predictions of the bootstrap models not trained on the i-th training sample (x_i, y_i) (Xu & Xie, 2020). This procedure inspires the EnCQR algorithm and, similarly, avoids overfitting, as shown by the results in Table 9.

The results above are interesting, but they do not explain the consistent underperformance of the TCNs compared to the LSTM networks. A second possible explanation is that the chosen convolutional hyperparameters, i.e. the kernel size, dilation factor, and resulting number of residual blocks, are not optimal for the forecasting problem at hand. The architecture of the TCNs and LSTM networks, i.e. the number of residual blocks and the number of LSTM layers used in the experiments, were fixed for all models and chosen beforehand. On the other hand, the optimal hyperparameter configuration, e.g. the learning rate, number of hidden units, and batch size, were for all models found via a hyperparameter search, providing the fairest comparison between the models. However, due to limited computational resources and time constraints, the size of the hyperparameter search space was limited. A more extensive search that includes more hyperparameters and network configurations could improve the performance of the models on the tasks at hand.

16.2.3 Linear vs. Non-linear Models

Neural networks are non-linear models, commonly termed black boxes (Dayhoff & DeLeo, 2001), as the interpretability of why a network predicts a specific value is not as straightforward as for linear models, such as the ones described in Section 6.5.1. For the latter models, one can draw one-to-one relationships between the parameters of the models and the variables in the data, whereas for non-linear models, this can not be done as easily. A neural network learns by being presented with a set of samples of observations, x, and labels y, where the network tries to approximate a function $f(x)$ that maps all observations into their corresponding label. The major advantage of neural networks is that they, in theory, can approximate any function, regardless of shape, without making any assumptions about the data generation process. As stated in Section 6.5.1, the advantages of the linear models are that they are easy to understand, interpret, and develop, but as their name suggests, these models assume that time series are generated from linear processes (Adebiyi et al., 2014), making them less suited for modeling complex non-linear problems.

The experimental results show that for the Portugal dataset, the neural network-based models generally outperform the SARIMA models in terms of validity, where only the SARIMA model for Station 90 produced comparable results. For the Elvia dataset, the differences in prediction interval coverage between the best performing neural network-based models and the SARIMA models are minor. Comparing the average coverage of the QRNNs and the coverage of the SARIMA models actually show that the latter have the highest coverage. Further, in terms of prediction interval sharpness, the SARIMA models produce the narrowest intervals for all end-users in the Elvia dataset. However, as discussed in Section 16.2.1, the results for the conformal prediction-based models for the Elvia dataset are not representative of the actual model performance due to the error made when selecting the hyperparameter configurations. If the hyperparameter search were performed correctly, one might expect that the conformal prediction-based models would perform similarly as for the Portugal dataset, however, this is not certain. Based on the knowledge that the latter models are not performing optimally, no general statements about the difference in performance between the models based on the results on the Elvia dataset are made.

Overall, the conformal prediction-based models outperform both the QRNN models and the SARIMA models for the Portugal time series. Seemingly, there is no significant difference in performance between the SARIMA and QRNN models, as they both generally tend to produce narrow prediction intervals, but the corresponding coverage is below the target coverage level.

16.2.4 Conformalization in the EnCQR Algorithm

In the EnCQR algorithm, similarly to the CQR algorithm, the prediction intervals constructed by the quantile regression ensemble learners are conformalized using an approach inspired by conformal prediction, where an error term is added to or subtracted from the interval's width. This error term quantifies the accuracy of the original interval, addressing both under- and overcoverage, and the intervals can be extended or shortened to improve both coverage and width. The effect of the conformalization step in the EnCQR algorithm is examined using the results from the EnCQR-LSTM model for Station 90 and Station 27 from the Portugal dataset, and the Industry and Household end-users in the Elvia dataset.

Table 10 shows that for the Elvia dataset, there is a considerable difference in prediction interval coverage and width between the original and conformalized intervals. The actual coverage of the former is significantly below the target coverage level, where the latter obtains valid coverage at 90%. For the Portugal dataset, the increase in prediction interval width of the conformalized intervals is relatively small, and the conformalization of the intervals results in valid coverage for both time series.

Table 10: Prediction interval average coverage and prediction interval normalized average width for the original and conformalized prediction intervals for the EnCQR-LSTM models.

Portugal dataset		
	PIAC/PINAW	
Time Series	conformalized interval	original interval
Station 90	0.904 / 0.283	0.836 / 0.238
Station 27	0.900 / 0.277	0.821 / 0.226
Elvia dataset		
	PIAC/PINAW	
Time Series	conformalized interval	original interval
Industry	0.954 / 0.464	0.479 / 0.197
Household	0.900 / 0.503	0.150 / 0.101

The original quantile regression prediction intervals and conformalized prediction intervals for a period of three days are shown in Fig. 38 and 39, where the improving effect of the conformalization is evident. For Station 27, the original intervals are unable to capture the peaks in electricity consumption because they are placed at too low load levels and, thus, consistently undercover. The conformalization extends the upper prediction interval bound, including the peaks, and shifts the lower bound upwards, improving the coverage at the cost of a slight increase in prediction interval width. The results for Station 90 are similar to Station 27, where the original interval initially was a bit too confident, resulting in coverage slightly below the target level.

As discussed in Section 16.2.1, the ensemble estimators are not able to satisfactorily model the true underlying model for the Elvia dataset, clearly visible for the Household time series in the middle plot in Fig. 39, resulting in large conformity scores, and the width of the prediction intervals therefore significantly increases. In terms of prediction interval coverage, the conformalized intervals successfully obtain valid marginal coverage, but the quality of the intervals is poor compared to the conformalized intervals for the Portugal time series. The bottom plot in Fig. 39 shows the prediction intervals constructed by the SARIMA model. Comparing the results in the middle and bottom plot, the figure clearly demonstrates the significantly increased prediction interval width of the conformalized intervals.

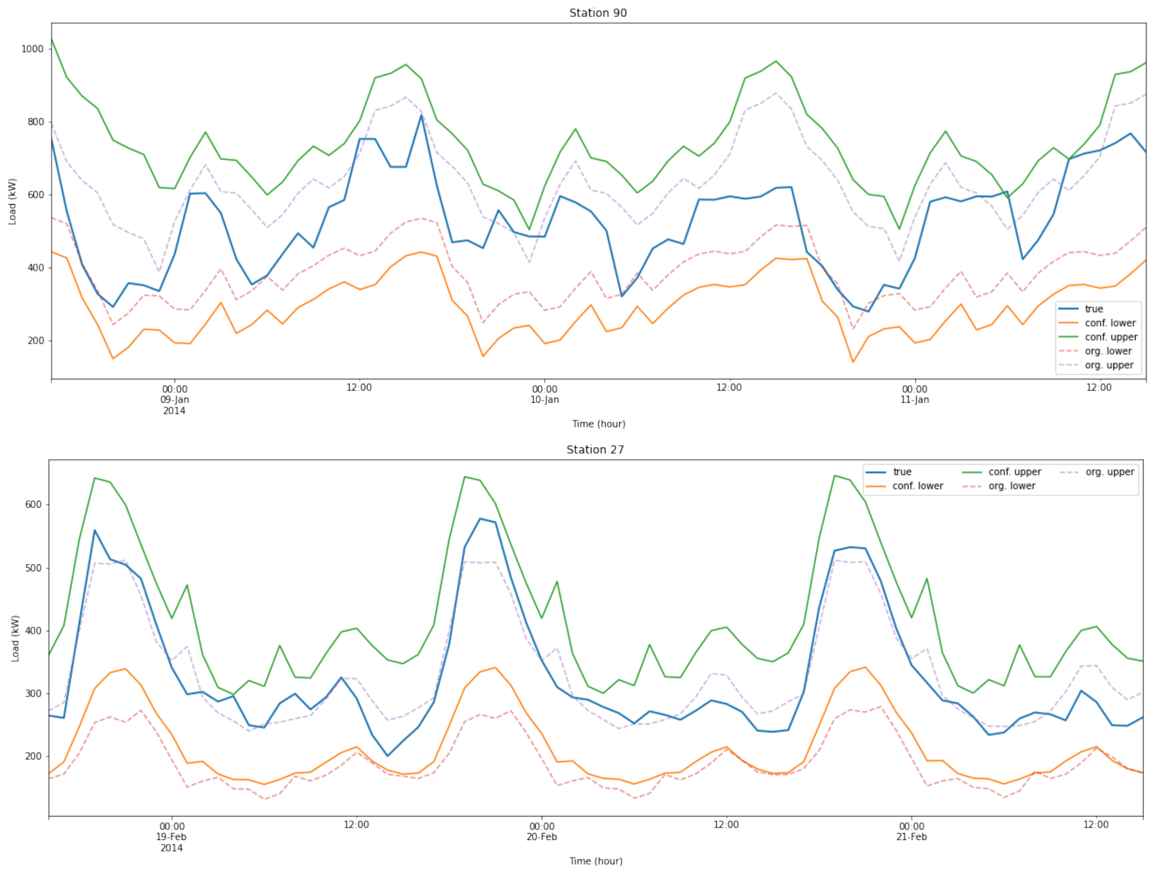


Figure 38: Original and conformalized prediction intervals for Station 90 and Station 27 from the Portugal dataset for a period of three days. The plots show that the coverage of the original intervals are for both time series improved by the conformalization. The dashed lines represent the original intervals.

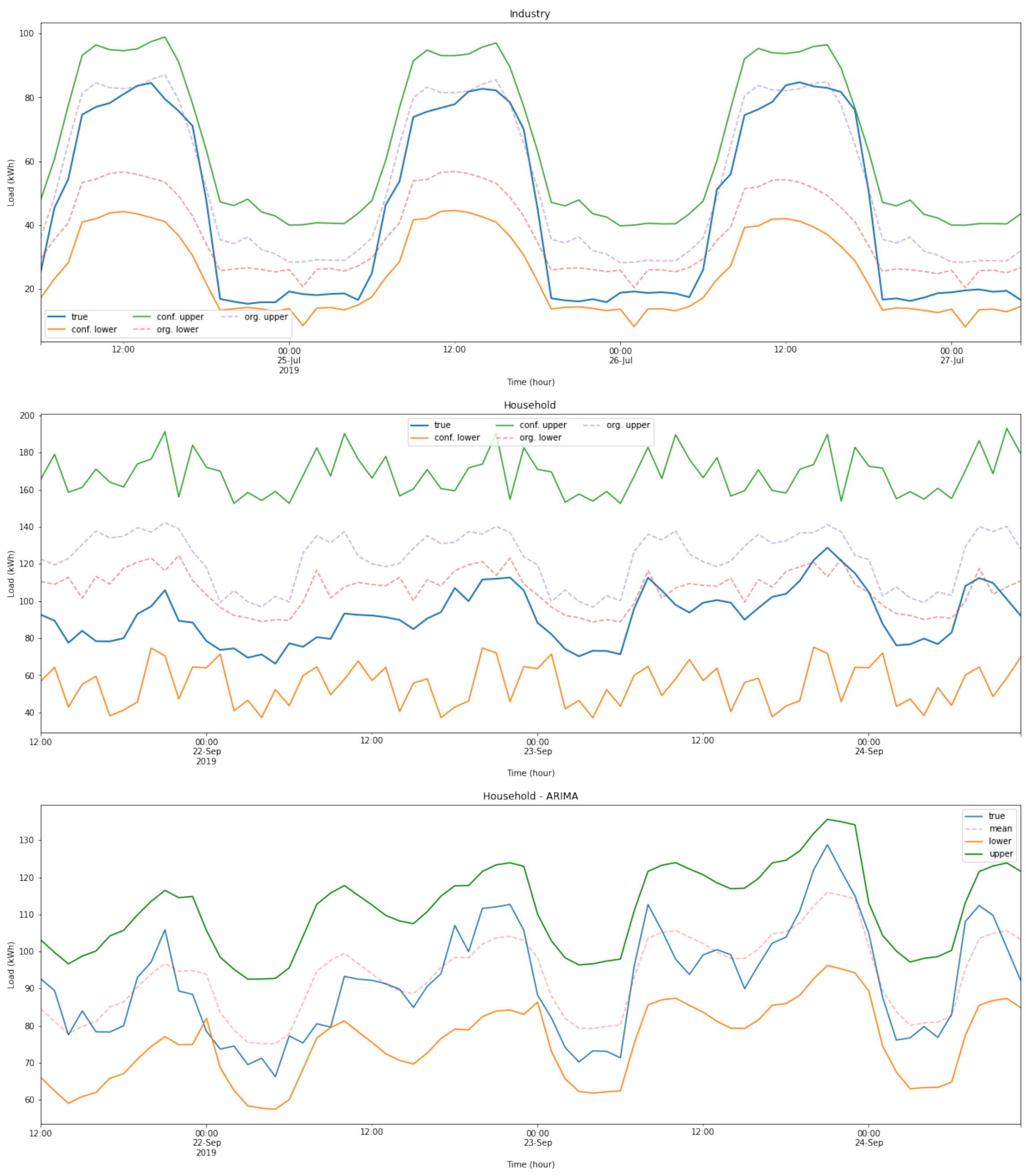


Figure 39: The two top plots show the original and conformalized prediction intervals for the Industry and Household time series from the Elvia dataset for a period of three days. These plots show that the coverage of the original intervals are for both time series improved by the conformalization. The dashed lines represent the original intervals. The bottom plot shows the prediction interval for Household time series constructed by the SARIMA model. Compared to the bottom plot, the middle plot shows clearly inflated prediction intervals.

Part V / Conclusions

This thesis studied the problem of probabilistic electricity load forecasting, focusing on producing marginally valid prediction intervals, and proposed a distribution-free probabilistic forecasting method that combines the advantages of conformal predictors and quantile regression into a single model. The experiments performed have shown that the proposed method can construct adaptive prediction intervals with approximately valid marginal coverage using two real-world electricity load datasets and two different underlying regression algorithms. The proposed method was compared with a traditional statistical time series forecasting method, a quantile regression-based method, and a conformal prediction-based method, where the two latter were implemented using neural networks, showing superior or comparable results. The experimental results show significant variations in performance between the methods, both for time series from the same dataset and between the two datasets. Overall, the following conclusions can be drawn:

- All conformal prediction-based models successfully constructed prediction intervals with approximately valid marginal coverage, as theory suggests they should. Marginally valid coverage was obtained even under misspecified models, demonstrating an important property and the strength of the proposed method.

- In terms of prediction interval sharpness, the proposed method generally produced the narrowest intervals, while providing the designed coverage level, proving that conformalized quantile regression-based models tend to outperform both standard conformal prediction-based models and quantile regression-based models in terms of prediction interval quality.

- The variation in the results between the individual network runs was inferior for the conformal prediction-based models compared to the quantile regression neural network models, indicating the robustness of these models.

- As base architecture to implement the proposed ensemble method, the long-short term memory (LSTM) architecture outperformed the temporal convolutional networks (TCN), producing higher quality prediction intervals on the two case-study considered in this thesis.

Further, the experimental results demonstrated that for the conformal prediction-based models, the neural network architecture should be tuned carefully to ensure that the ensemble learners are able to model the true underlying data-generating model successfully. However, as shown in the experimental results, valid marginal can be obtained regardless of the accuracy of the predictions constructed by ensemble estimators, at the expense of prediction interval quality, which become more inflated and less informative.

Future work can include the following directions: (1) Find an optimal neural network architecture, which can improve the prediction of the time series in the multivariate dataset. In return, this will allow to exploit the proposed method at its full potential; (2) Incorporate post-processing of the quantile prediction to eliminate quantile crossing, a problem that occurs when the lower quantile estimate is larger than the upper quantile estimate (Romano, Patterson, & Candès, 2019); (3) Explore the performance of the proposed method in the presence of missing data (Xu & Xie, 2020; Bianchi, Livi, Mikalsen, Kampffmeyer, & Jenssen, 2019); (4) Explore the possibility of constructing a loss function that targets prediction interval coverage and sharpness separately, to obtain a more controlled trade-off between the two (Chung, Neiswanger, Char, & Schneider, 2020).

References

Abdel-Hamid, O., Mohamed, A.-r., Jiang, H., Deng, L., Penn, G., & Yu, D. (2014). Convolutional neural networks for speech recognition. *IEEE/ACM Transactions on audio, speech, and language processing*, *22*(10), 1533–1545.

Adebiyi, A. A., Adewumi, A. O., & Ayo, C. K. (2014). Comparison of arima and artificial neural networks models for stock price prediction. *Journal of Applied Mathematics*, *2014*.

Adhikari, R., & Agrawal, R. K. (2013). An introductory study on time series modeling and forecasting. *arXiv preprint arXiv:1302.6613*.

Almeshaiei, E., & Soltan, H. (2011). A methodology for electric power load forecasting. *Alexandria Engineering Journal*, *50*(2), 137–144.

Anthony, M., & Bartlett, P. L. (2009). *Neural network learning: Theoretical foundations.* cambridge university press.

Bai, S., Kolter, J. Z., & Koltun, V. (2018). An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271*.

Barber, R. F., Candes, E. J., Ramdas, A., & Tibshirani, R. J. (2019). The limits of distribution-free conditional predictive inference. *arXiv preprint arXiv:1903.04684*.

Bengio, Y. (2009). *Learning deep architectures for ai.* Now Publishers Inc.

Bernardo, J. M. (1996). The concept of exchangeability and its applications. *Far East Journal of Mathematical Sciences*, *4*, 111–122.

Bianchi, F. M., Livi, L., Mikalsen, K. Ø., Kampffmeyer, M., & Jenssen, R. (2019). Learning representations of multivariate time series with missing data. *Pattern Recognition*, *96*, 106973.

Bianchi, F. M., Maiorino, E., Kampffmeyer, M. C., Rizzi, A., & Jenssen, R. (2017). *An overview and comparative analysis of recurrent neural networks for short term load forecasting.* Springer International Publishing.

Borovykh, A., Bohte, S., & Oosterlee, C. W. (2017). Conditional time series forecasting with convolutional neural networks. *arXiv preprint arXiv:1703.04691*.

Box, G. E., Jenkins, G. M., Reinsel, G. C., & Ljung, G. M. (2015). *Time series analysis: forecasting and control.* John Wiley & Sons.

Brockwell, P. J., Brockwell, P. J., Davis, R. A., & Davis, R. A. (2016). *Introduction to time series and forecasting.* Springer.

Brownlee, J. (2018). *Deep learning for time series forecasting: Predict the future with mlps, cnns and lstms in python.* Machine Learning Mastery.

Caron, M., Bojanowski, P., Joulin, A., & Douze, M. (2018). Deep clustering for unsupervised learning of visual features. In *Proceedings of the european conference on computer vision (eccv)* (pp. 132–149).

Casella, G., & Berger, R. L. (2021). *Statistical inference.* Cengage Learning.

Chen, Y., Kang, Y., Chen, Y., & Wang, Z. (2020). Probabilistic forecasting with temporal convolutional neural network. *Neurocomputing*, *399*, 491 - 501.

Chollet, F. (2015). *Keras.* `https://keras.io`.

Chung, Y., Neiswanger, W., Char, I., & Schneider, J. (2020). Beyond pinball loss: Quantile methods for calibrated uncertainty quantification. *arXiv preprint arXiv:2011.09588*.

Dalal, N., Mølnå, M., Herrem, M., Røen, M., & Gundersen, O. E. (2020). Day-ahead forecasting of losses in the distribution network. In *Proceedings of the aaai conference on artificial intelligence* (Vol. 34, pp. 13148–13155).

Dang-Ha, T., Bianchi, F. M., & Olsson, R. (2017). Local short term electricity load forecasting: Automatic approaches. In *2017 international joint conference on neural networks (ijcnn)* (p. 4267-4274). doi: 10.1109/IJCNN.2017.7966396

Dayhoff, J. E., & DeLeo, J. M. (2001). Artificial neural networks: opening the black box. *Cancer: Interdisciplinary International Journal of the American Cancer Society*, *91*(S8), 1615–1635.

De Gooijer, J. G., & Hyndman, R. J. (2006). 25 years of time series forecasting. *International Journal of Forecasting*, *22*(3), 443-473. Retrieved from `https://www.sciencedirect.com/science/article/pii/S0169207006000021` (Twenty five years of forecasting) doi: https://doi.org/10.1016/j.ijforecast.2006.01.001

Dickey, D. A., & Fuller, W. A. (1979). Distribution of the estimators for autoregressive time series with a unit root. *Journal of the American statistical association*, *74*(366a), 427–431.

Dutilleul, P., & Legendre, P. (1993). Spatial heterogeneity against heteroscedasticity: an ecological paradigm versus a statistical concept. *Oikos*, 152–171.

Elvers, A., Voß, M., & Albayrak, S. (2019). Short-term probabilistic load forecasting at low aggregation levels using convolutional neural networks. In *2019 ieee milan powertech* (p. 1-6). doi: 10.1109/PTC.2019.8810811

Gammerman A., V. V., Vovk V. (1998). Learning by transduction. *Proceedings of the fourteenth conference on uncertainty in artificial intelligence*, 48-155.

Gasparin, A., Lukovic, S., & Alippi, C. (2019). Deep learning for time series forecasting: The electric load case. *arXiv preprint arXiv:1907.09207*.

Gasthaus, J., Benidis, K., Wang, Y., Rangapuram, S. S., Salinas, D., Flunkert, V., & Januschowski, T. (2019). Probabilistic forecasting with spline quantile function rnns. In *The 22nd international conference on artificial intelligence and statistics* (pp. 1901–1910).

Gilchrist, W. (2000). *Statistical modelling with quantile functions.* CRC Press.

Gneiting, T., Balabdaoui, F., & Raftery, A. E. (2007). Probabilistic forecasts, calibration and sharpness. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, *69*(2), 243–268.

Gneiting, T., & Katzfuss, M. (2014). Probabilistic forecasting. *Annual Review of Statistics and Its Application*, *1*, 125–151.

Gneiting, T., & Raftery, A. E. (2007). Strictly proper scoring rules, prediction, and estimation. *Journal of the American statistical Association*, *102*(477), 359–378.

Gonzalez, R., & Woods, R. (2018). *Digital image processing 4. edition.* Pearson.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning.* MIT Press. (`http://www.deeplearningbook.org`)

Grushka-Cockayne, Y., Lichtendahl Jr, K. C., Jose, V. R. R., & Winkler, R. L. (2017). Quantile evaluation, sensitivity to bracketing, and sharing business payoffs. *Operations Research*, *65*(3), 712–728.

Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The elements of statistical learning: data mining, inference, and prediction.* Springer Science & Business Media.

Hatalis, K., Lamadrid, A. J., Scheinberg, K., & Kishore, S. (2017). Smooth pinball neural network for probabilistic forecasting of wind power. *arXiv preprint arXiv:1710.01720*.

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *CoRR*, *abs/1502.01852*. Retrieved from `http://arxiv.org/abs/1502.01852`

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 770–778).

Hong, T., & Fan, S. (2016). Probabilistic electric load forecasting: A tutorial review. *International Journal of Forecasting*, *32*(3), 914–938.

Hong, T., Pinson, P., Fan, S., Zareipour, H., Troccoli, A., & Hyndman, R. J. (2016). *Probabilistic energy forecasting: Global energy forecasting competition 2014 and beyond.* Elsevier.

Hornik, K., Stinchcombe, M., & White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, *2*, 359-366.

Huang, F., Xie, G., & Xiao, R. (2009). Research on ensemble learning. In *2009 international conference on artificial intelligence and computational intelligence* (Vol. 3, pp. 249–252).

Hyndman, R. J., & Athanasopoulos, G. (2018). *Forecasting: principles and practice.* OTexts.

Infield, D., & Freris, L. (2009). *Renewable energy in power systems.* John Wiley & Sons.

Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*.

Jordan, M. I., & Mitchell, T. M. (2015). Machine learning: Trends, perspectives, and prospects. *Science*, *349*(6245), 255–260.

Kath, C., & Ziel, F. (2020). Conformal prediction interval estimation and applications to day-ahead and intraday power markets. *International Journal of Forecasting*.

Keren, G., Cummins, N., & Schuller, B. (2018). Calibrated prediction intervals for neural network regressors. *IEEE Access*, *6*, 54033–54041.

Kim, B., Xu, C., & Barber, R. F. (2020). Predictive inference is free with the jackknife+-after-bootstrap. *arXiv preprint arXiv:2002.09025*.

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Koenker, R., & Bassett Jr, G. (1978). Regression quantiles. *Econometrica: journal of the Econometric Society*, 33–50.

Lea, C., Flynn, M. D., Vidal, R., Reiter, A., & Hager, G. D. (2017). Temporal convolutional networks for action segmentation and detection. In *proceedings of the ieee conference on computer vision and pattern recognition* (pp. 156–165).

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., & Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural computation*, *1*(4), 541–551.

Liu, B., Nowotarski, J., Hong, T., & Weron, R. (2017). Probabilistic load forecasting via quantile regression averaging on sister forecasts. *IEEE Transactions on Smart Grid*, *8*(2), 730-737.

Luo, W., Li, Y., Urtasun, R., & Zemel, R. (2017). Understanding the effective receptive field in deep convolutional neural networks. *arXiv preprint arXiv:1701.04128*.

Makridakis, S., Spiliotis, E., & Assimakopoulos, V. (2018). Statistical and machine learning forecasting methods: Concerns and ways forward. *PloS one*, *13*(3), e0194889.

Masum, S., Liu, Y., & Chiverton, J. (2018). Multi-step time series forecasting of electric load using machine learning models. In *International conference on artificial* *intelligence and soft computing* (pp. 148–159).

McGill, R., Tukey, J. W., & Larsen, W. A. (1978). Variations of box plots. *The American Statistician*, *32*(1), 12–16.

Nowotarski, J., & Weron, R. (2015). Computing electricity spot price prediction intervals using quantile regression and forecast averaging. *Computational Statistics*, *30*(3), 791–803.

Nowotarski, J., & Weron, R. (2018). Recent advances in electricity price forecasting: A review of probabilistic forecasting. *Renewable and Sustainable Energy Reviews*, *81*, 1548–1568.

Oord, A. v. d., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., ... Kavukcuoglu, K. (2016). Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*.

Papadopoulos, H. (2008). Inductive conformal prediction: Theory and application to neural networks. In *Tools in artificial intelligence.* Citeseer.

Papadopoulos, H., Proedrou, K., Vovk, V., & Gammerman, A. (2002). Inductive confidence machines for regression. In *European conference on machine learning* (pp. 345–356).

Pascanu, R., Gulcehre, C., Cho, K., & Bengio, Y. (2013). How to construct deep recurrent neural networks. *arXiv preprint arXiv:1312.6026*.

Quan, H., Srinivasan, D., & Khosravi, A. (2013). Short-term load and wind power forecasting using neural network-based prediction intervals. *IEEE transactions on neural networks and learning systems*, *25*(2), 303–315.

Romano, Y., Barber, R. F., Sabatti, C., & Candès, E. J. (2019). With malice towards none: Assessing uncertainty via equalized coverage. *arXiv preprint arXiv:1908.05428*.

Romano, Y., Patterson, E., & Candès, E. J. (2019). Conformalized quantile regression. *arXiv preprint arXiv:1905.03222*.

Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, *65*(6), 386.

Rosenblatt, M. (1956). A central limit theorem and a strong mixing condition. *Proceedings of the National Academy of Sciences of the United States of America*, *42*(1), 43.

Sesia, M., & Candès, E. J. (2020). A comparison of some conformal quantile regression methods. *Stat*, *9*(1), e261.

Shafer, G., & Vovk, V. (2008). A tutorial on conformal prediction. *Journal of Machine Learning Research*, *9*(3).

Shepero, M., Van Der Meer, D., Munkhammar, J., & Widén, J. (2018). Residential probabilistic load forecasting: A method using gaussian process designed for electric load data. *Applied Energy*, *218*, 159–172.

Shi, H., Worden, K., & Cross, E. J. (2019). A cointegration approach for heteroscedastic data based on a time series decomposition: an application to structural health monitoring. *Mechanical Systems and Signal Processing*, *120*, 16–31.

Shumway, R. H., & Stoffer, D. S. (2017). *Time series analysis and its applications: with r examples.* Springer.

Taieb, S. B., Bontempi, G., Atiya, A. F., & 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.

Taylor, J. W. (2000). A quantile regression neural network approach to estimating the conditional density of multiperiod returns. *Journal of Forecasting*, *19*(4), 299–311.

Taylor, J. W., & McSharry, P. E. (2007). Short-term load forecasting methods: An evaluation based on european data. *IEEE Transactions on Power Systems*, *22*(4), 2213-2219.

Terrell, G. R., & Scott, D. W. (1992). Variable kernel density estimation. *The Annals of Statistics*, 1236–1265.

Theodoridis, S., & Koutroumbas, K. (2009). *Pattern recognition* (4th ed. ed.). Elsevier.

Tsay, R. S. (2014). *Multivariate time series analysis with r and financial applications.* Wiley.

Vovk, V., Gammerman, A., & Shafer, G. (2005). *Algorithmic learning in a random world.* Berlin, Heidelberg: Springer-Verlag.

Wang, Y., Gan, D., Sun, M., Zhang, N., Lu, Z., & Kang, C. (2019). Probabilistic individual load forecasting using pinball loss guided lstm. *Applied Energy*, *235*, 10–20.

Wang, Y., Zhang, N., Tan, Y., Hong, T., Kirschen, D. S., & Kang, C. (2019). Combining probabilistic load forecasts. *IEEE Transactions on Smart Grid*, *10*(4), 3664-3674. doi: 10.1109/TSG.2018.2833869

Wen, R., Torkkola, K., Narayanaswamy, B., & Madeka, D. (2017). A multi-horizon quantile recurrent forecaster. *arXiv preprint arXiv:1711.11053*.

Wolpert, D. H. (1996). The lack of a priori distinctions between learning algorithms. *Neural computation*, *8*(7), 1341–1390.

Xu, C., & Xie, Y. (2020). Conformal prediction interval for dynamic time-series. *arXiv preprint arXiv:2010.09107*.

Yang, Y., Wu, J., Chen, Y., & Li, C. (2013). A new strategy for short-term load forecasting. In *Abstract and applied analysis* (Vol. 2013).

Yule, G. U. (1927). On a method of investigating periodicities in disturbed series, with special reference to wolfer's sunspot numbers. *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character*, *226*, 267–298. Retrieved from `http://www.jstor.org/stable/91170`

Zeni, G., Fontana, M., & Vantini, S. (2020). Conformal prediction: a unified review of theory and new challenges. *arXiv preprint arXiv:2005.07972*.

Zhang, C., & Ma, Y. (2012). *Ensemble machine learning: methods and applications.* Springer.

Zhang, G. P. (2001). An investigation of neural networks for linear time-series forecasting. *Computers & Operations Research*, *28*(12), 1183–1202.

Zhou, Z.-H. (2012). *Ensemble methods: foundations and algorithms.* CRC press.