UNIVERSITÄT ZU LÜBECK
INSTITUTE FOR SOFTWARE ENGINEERING
AND PROGRAMMING LANGUAGES

Machine Learning for Time Series Prediction of Energy Data

Maschinelles Lernen für die Zeitreihenprognose von Energiedaten

Bachelorarbeit

verfasst am
Institut für Softwaretechnik und Programmiersprachen

im Rahmen des Studiengangs
Informatik
der Universität zu Lübeck

vorgelegt von
Christopher Walther

ausgegeben und betreut von
Prof. Dr. Martin Leucker

mit Unterstützung von
Dr. Martin Sachenbacher
Daniel Thoma

Lübeck, den 2. November 2021

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt, dass ich diese Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

_____

Christopher Walther

Zusammenfassung

Der wachsende Anteil an erneuerbaren Energiequellen in der Stromerzeugung gefährdet die Stabilität des Stromnetzes. Der Ausgleich von Lastspitzen durch die gezielte Rückspeisung von Energie aus Batterien in Elektrofahrzeugen kann Abhilfe schaffen. Das ReNuBiL Projekt (Reallabor Nutzerorientiertes Bidirektionales Laden) erforscht das bidirektionale Laden von Elektrofahrzeugen und die Planung von deren Buchungen basierend auf Zeitreihenvorhersagen von Energiedaten und dem Bedarf an Netzinteraktion. In dieser Arbeit werden drei auf künstlichen neuronalen Netzen basierende Modelle des maschinellen Lernens vorgestellt, die aus einem einfachen, vollständig verbundenen Netz, einem faltenden Netz und einem autoregressiven LSTM-Netz (Long Short-Term Memory) bestehen. Die drei Energiezeitreihen werden jeweils einen Tag in die Zukunft vorhergesagt und umfassen den Strompreis, den Stromverbrauch des Audimax Gebäudes der Universität zu Lübeck und die erneuerbare Stromerzeugung. Die Modelle werden mit TensorFlow und Keras unter Verwendung von Hardware-Beschleunigung trainiert, unter anderem mit der NVIDIA DGX2 im AI Lab Lübeck. Eine Verbesserung der Vorhersagegenauigkeit von mindestens 54% im Vergleich zu statistischen Leistungsmaßstäben wurde für alle drei Zeitreihen unter Verwendung von optimal konfigurierten Modellen erreicht. Vorverarbeitungsschritte, wie das Hinzufügen von Temperaturdaten und die saisonale Anpassung mit Hilfe von Medianfenstern, wurden getestet. Dabei wurde festgestellt, dass letztere Methode eine Verbesserung herbeiführte. Zusätzlich wird ein Ansatz für qualitative Vorhersagen getestet. Abschließend macht diese Arbeit Vorschläge zur Verbesserung der Modelle für die zukünftige Forschung.

Abstract

The rise of renewable energy sources threatens the power grid's stability. This can be remedied by discharging batteries in electric vehicles back into the grid at the right moment to compensate for peak loads. The ReNuBiL project (Reallabor Nutzerorientiertes Bidirektionales Laden, English: real-world laboratory for user-oriented bidirectional charging) explores bidirectional charging of electric vehicles and the scheduling of bookings based on time series predictions of energy data and the need for grid interaction. This work presents three artificial neural network-based machine learning models such as a basic dense neural network model, a convolution model and an autoregressive Long Short-Term Memory (LSTM) model that predict three energy time series one day into the future. The predicted time series comprise the electricity price, the power consumption of the Audimax building of the University of Lübeck and the renewable power generation. The models are trained with TensorFlow and Keras using hardware acceleration, such as the NVIDIA DGX2 machine at the AI Lab Lübeck. An improvement in performance of at least 54% compared to statistical baselines was achieved for all three time series using the models in optimal configurations. Preprocessing steps such as adding temperature data and seasonal adjustment using median windowing are tested and the latter is found to improve performance. A qualitative prediction approach is also evaluated. Finally, this work offers suggestions on improving these models in future research.

Acknowledgements

I would like to thank Dr. Martin Sachenbacher and Daniel Thoma for frequently giving me advice and supporting me, as well as the ReNuBiL team for providing the opportunity and resources for this work.

I would also like to express my gratitude towards my parents and friends who always supported and encouraged me as I was writing this thesis. Special thanks go to my mother whose tireless proofreading was indispensable.

Contents

1

Introduction

1.1 Motivation

As we grow increasingly reliant on renewable energy sources, the danger of a fluctuating power output looms over efforts to move into the age of a carbon-neutral power grid. Many renewable energy sources harness the power of the sun, wind or tides for generating electrical power. While they are generally cheap and environmentally friendly, their biggest disadvantage is that their output changes depending on external factors, which makes them hard to integrate into a grid system at scale. Additionally, extreme weather events can cause the demand for energy to rise suddenly and disrupt all types of power generation as was seen in the 2021 blackout in Texas, USA [6]. The simultaneous need for carbon-neutral energy and a grid resilient to the impacts of climate change on power generation [37] heightens the importance of technologies capable of stabilizing power grids.

A data-driven approach to grid stabilization is shaped by the available data sources. As such, a number of sources are considered. Firstly, electricity price data introduces economic aspects of the power grid. Secondly, power consumption data of a single building gives insight into the patterns of energy usage and the effects of human activity on it. And finally, power generation, in particular renewable power generation, allows for the study of both sides of a power grid from a climate-conscious perspective.

The energy output of renewable sources is heavily influenced by environmental factors, many of which are determined by very complex systems. Wind speed, solar irradiance, precipitation and sea currents are examples of factors important to renewable power generation that are determined by the weather and climate. Power consumption, on the other hand, is largely determined by human activity but is just as difficult to model accurately, as it is also the product of complex and random interactions.

Through machine learning, computers can become adept at tasks that are otherwise difficult or even impossible to program explicitly. This presents an opportunity for modeling the complex systems that need to be understood in order to balance and improve the power grid. This work explores whether machine learning, and artificial neural networks in particular, can be used to effectively model parts of these systems and make predictions of the associated time series data. Such predictions are used in the context of the

ReNuBiL project [35] (Reallabor Nutzerorientiertes Bidirektionales Laden, English: real-world laboratory for user-oriented bidirectional charging) for compensating peak loads using electric vehicles on the campus of the University of Lübeck. The project explores the technological and psychological aspects of grid stabilization through a small-scale implementation of intelligent bidirectional charging.

1.2 Contributions of this Thesis

In this work various machine learning models for time series prediction are applied to electricity price, power consumption and power generation data. Choosing the best model type and configuration for a particular application can be difficult but is nonetheless an important factor for performance. Due to the complex processes that produce this data, a very good prediction or a prediction far into the future is not achieved. Rather a prediction with acceptable accuracy one day into the future can be produced with these models. Simpler dense *artificial neural network* (ANN, or NN for short) models and convolutional ANN models produced the best results while an autoregressive *Long Short-Term Memory* (LSTM) model was not far behind. Detailed experiments were conducted to determine the best methods and models for each data source. In order to verify the effectiveness of these machine learning-based models, their performance is compared to baselines given by simple statistical methods.

Data was collected from three different sites. Forecasts of the electricity prices using data from a local utility company are made, as they could be useful in lowering energy costs. Secondly, the power consumption of the Audimax of the University of Lübeck is predicted so that load spikes may be compensated. And finally, a prediction of the renewable energy generation volume in Germany is explored, in order to be able to give preference to charging the electric vehicles with predominantly renewable energy. Changes in the structure of the Audimax power consumption throughout the spring and summer of 2021 posed an additional challenge but also made the dataset more diverse.

All three data sources naturally contain cyclic patterns relating to a change in power consumption or renewable energy availability over the course of a day. Seasonal adjustment is a preprocessing step that removes these patterns from the data and allows models to predict the non-seasonal parts of a time series. It is found to be effective at improving model performance in most cases. The combination of these techniques and additionally training models to predict the difference between one day and the previous one also produces good results. The best models made predictions that had an at least 54% lower error than the statistical baselines.

Since, ultimately, scheduling decisions will be based on the prediction of these time series, a qualitative approach is presented in which various metrics of the data, such as the spread or minimum and maximum, are predicted in favor of the data points themselves. Giving models access to air temperature data is also tested but it does not appear to make the predictions better, however, because the system is probably too complex for this type of model to properly comprehend.

A Python script handles the preparation of the training data, inputting additional data sources, tuning and training models, outputting the results and optionally also plot-

ting them. The models are created with the help of Google's TensorFlow with the Keras API, while Pandas is used for processing the data. Training is done using GPUs and is optionally further accelerated by using the NVIDIA DGX2 machine available at the AI Lab Lübeck.

1.3 Related Work

Due to the importance of the prediction of energy prices, consumption, and generation to infrastructure providers there is a lot of literature available on this topic. Amasyali and El-Gohary provide an overview [4] of the methods used in the prediction of building energy consumption. Artificial neural networks trained using back propagation (BPNN) [16] like the models used in this work are, in general, found to be a common and effective tool for this type of time series prediction.

Forecasts of electricity prices using LSTM neural networks have been performed with a wavelet-transform [46] preprocessing stage [8, 28] as well as more complex decomposition and model structures [49]. Determining the optimal hyperparameters for a network remains a central challenge in this field. Peng et al. present a novel evolutionary algorithm called differential evolution, which finds suitable hyperparameters for an LSTM model and allows it to outperform existing models [32]. Another approach to choosing an effective LSTM architecture and model inputs is proposed by Gundu and Simon, which uses a particle swarm optimization technique [14].

Power consumption, or the *load*, of a grid as a whole is interesting to forecast as it provides crucial guidance to grid operators who need to match it with generation capacities. A review of different established methods, such as engineering, statistical and machine learning methods, for the prediction of power consumption is provided by Zhao and Magoulès [48]. Pramono et al. present a method for short-term load forecasting using models that combine *convolutional neural networks* (CNNs) and LSTM networks [33]. Similar methods can also be applied to individual households [23] and industrial systems [43].

Work on forecasts of energy generation focuses on solar and wind energy as these are the most widespread energy sources that fluctuate throughout the day and year. Solar irradiance, the intensity of the sunlight hitting the earth, alongside other factors determining photovoltaic power output is heavily influenced by climate conditions and the weather. The solar irradiance correlates to photovoltaic output and therefore one approach is to make predictions of the solar irradiance based on existing weather forecasts [34, 45]. Including a weather forecast in the problem the model has to solve can make producing a good model even more challenging since weather forecasting is a difficult problem in itself. Weather forecasts using large-scale simulations and specialized machine learning techniques already provide sources of weather parameters for all geographical positions. Wavelet decomposition as a preprocessing step also gives good results [7] according to Cao and Cao.

Given the relatively direct influence of the weather on solar irradiance, a more traditional approach that simulates the irradiation of photovoltaic power plants is successful as well [26]. Aggarwal and Saini reviewed a number of other model types apart from

artificial neural networks and found a combination of model types to perform best [2]. Kumar et al. apply forecasts using LSTM models of wind and solar power generation to small grid systems in which the fluctuations of renewable energy sources can have an even greater effect [22]. While there are methods utilizing physical simulations and statistical models for wind power forecasts [5, 36, 15], short-term predictions of wind power using LSTM models with a wavelet transform were found to be more accurate [25]. As with the forecasts of energy prices, the problem of optimizing an LSTM model's hyperparameters remains difficult but can be tamed by using, for example, a genetic algorithm as demonstrated by Shahid et al. [38].

A common strategy in all three areas is to use CNNs alone or in combination with LSTM networks [44, 21, 20, 33, 23] because they are particularly well suited for learning and extracting patterns. However, Zhang and Kline also found simple models to work well if seasonal adjustment is utilized [47].

1.4 Structure of this Thesis

This section describes the order and content of the following chapters and sections. In general, the formal structures are discussed first followed by specific implementation details and finally the experimental results.

A formal basis for time series prediction and the evaluation of such predictions using error metrics is established in chapter 2. It also introduces simple statistical methods that are used a baseline against which sophisticated time series prediction models are benchmarked. Additionally, three types of artificial neural network models are described and illustrated with diagrams. In chapter 3 the application context for the time series prediction task in the ReNuBiL project is presented and its requirements are characterized. The three data sources are introduced at the level of the application context. Chapter 4 describes the technical details of the data sources and then goes into the various methods with which the data is preprocessed. The methods include adding time signals and air temperature data, downsampling and normalization of the data, and optionally applying seasonal adjustment using median windowing. It also introduces the concept of qualitative prediction.

Before model training can begin, sequences of inputs and labels have to be generated. This is described in chapter 5, which then also gives the performance results of the statistical baselines and describes the no input baseline. The process of hyperparameter tuning is described alongside an overview of each models' hyperparameters. The Evaluation chapter 6 starts off with implementation details after which the methodology of the evaluation is described in detail. Then, the tuned hyperparameters of the models are given and finally the performance of each model is evaluated. This chapter also goes into how well each of the methods themselves worked. The last chapter 7 summarizes this work and its results, followed by a detailed outlook of the next steps and the opportunities for future work.

2

Time Series Prediction

2.1 Problem Definition

A sequence of vectors dependent on time t serves as a basis for modeling the time series prediction task. This section is adapted from parts of section 2.1 of Dorffner's overview of using neural networks for time series processing [12]. Such a sequence can be represented as a function $\vec{x} : \mathbb{R}_{\geq 0} \to \mathbb{R}^k$ that yields k-dimensional vectors.

As x represents real-world data, it is continuous with respect to the time variable t. However, since measurements, also called *samples*, can only be taken at discrete time steps the whole function is not available. The samples are taken at regular intervals determined by the sampling interval $1/f$ where f is the sampling frequency. Because the sampling process cannot be perfect, the samples are taken at slightly irregular sampling times $t_0, t_1, t_2, \ldots$ from which values at exactly spaced sampling times $t \in \mathbb{N}_0$ are later interpolated. This preprocessing step happens before the prediction stage and, in the following, is assumed to have already been performed.

For the task of predicting $\vec{y}(t+d)$ using n vectors from $\vec{x}$, a function $\mathbf{F} : \mathbb{R}^{k \times n} \to \mathbb{R}^h$ is required. The time series $\vec{y} : \mathbb{R}_{\geq 0} \to \mathbb{R}^h$ being predicted yields h-dimensional vectors since not all values in vectors of $\vec{x}$ are interesting to predict but can be useful in making an accurate prediction. These vectors are referred to as *labels* in a prediction task. The prediction estimates $\vec{y}$ as $\hat{\vec{y}}$:

$$\hat{\vec{y}}(t+d) = \mathbf{F}(\vec{x}(t), \vec{x}(t-1), \ldots, \vec{x}(t-(n-1)))$$

In this case additional secondary values that can be useful for some models are included in $\vec{x}$. d refers to the prediction *lag* and is in this work 1 since predictions starting at the current moment are desired. The function $\mathbf{F}$ can be considered a *model* of the time series as its inputs can be successively replaced with previous predictions making the prediction of multiple samples possible. As such, it is capable of generating the series from a starting set of values. Obviously $\mathbf{F}$ is only useful when it has less input parameters than there are elements in the time series. In practice, $\mathbf{F}$ will always receive the last n elements from $\vec{x}$ forming a window of a certain time span.

2.2 Error Metrics

With $\vec{y}$ being the unknown value that would be observed in the future and $\mathbf{F}$ approximating it with $\hat{\vec{y}}$, the forecast is evaluated by computing the mean error $E_{\mathbf{F}}$ over a test time series of length m:

$$E_{\mathbf{F}} = \frac{1}{m} \sum_{i=0}^{m-1} e(\hat{\vec{y}}(t-i), \vec{y}(t-i))$$

It is denoted $E_{\mathbf{F}}$ to emphasize that the performance of $\mathbf{F}$ is being calculated. The error between the forecasted value and the real sequence element is measured by the error function e. The goal is to minimize $E_{\mathbf{F}}$ to an acceptable level. The largest acceptable error is dependent on the error metric and the context. The two metrics used here are the *mean squared error* (MSE) and the *mean absolute error* (MAE). MSE is the error metric $E_{\mathbf{F}}$ using the squared error e_s, which is calculated as follows:

$$\mathrm{e_s}(\hat{\vec{v}}, \vec{v}) = (\hat{\vec{v}} - \vec{v})^2$$

MAE, on the other hand, uses the absolute error e_a for a proportional error response:

$$\mathrm{e_a}(\hat{\vec{v}}, \vec{v}) = |\hat{\vec{v}} - \vec{v}|$$

MSE particularly emphasizes large errors while MAE weights all errors proportional to their magnitude. The choice of error metric affects the behavior of a model since the training process optimizes it to minimize the error by adjusting the model's internal parameters. If avoiding large spikes is very important, MSE is a good choice. If large spikes are just as important to consider as a consistent offset of the prediction from the correct values over a longer period of time, MAE should be chosen. This means picking an error metric is actually part of specifying the problem the model is supposed to solve since the error metric determines how well the problem was solved and to what degree the specification was fulfilled.

2.3 Statistical Methods

A more accurate assessment of a model's performance can be made if it is compared to a simple baseline model. If a complicated model is not better than the simple baseline model then it can be discarded on the basis of being unnecessarily complicated. The baseline models make rough estimates of the course of the data over the prediction interval.

The simplest baseline repeats the latest input value for the entire output. In formal terms this is represented by the model $\mathbf{F}_{\text{last}}$:

$$\mathbf{F}_{\text{last}}(\vec{x}(t), \ldots, \vec{x}(t-(n-1))) = \vec{x}(t)$$

Another simple baseline repeats the entire input as the output and is given with the model $\mathbf{F}_{\text{repeat}}$:

$$\mathbf{F}_{\text{repeat}}(\vec{x}(t), \ldots, \vec{x}(t-(n-1))) = \vec{x}(t-(n-1))$$

For simplicity $\vec{x}$ is identical to $\vec{y}$ here. In practice, an identity model would select some values from each element in $\vec{x}$. Figure 2.1 illustrates how the two simple baselines work using power consumption data.

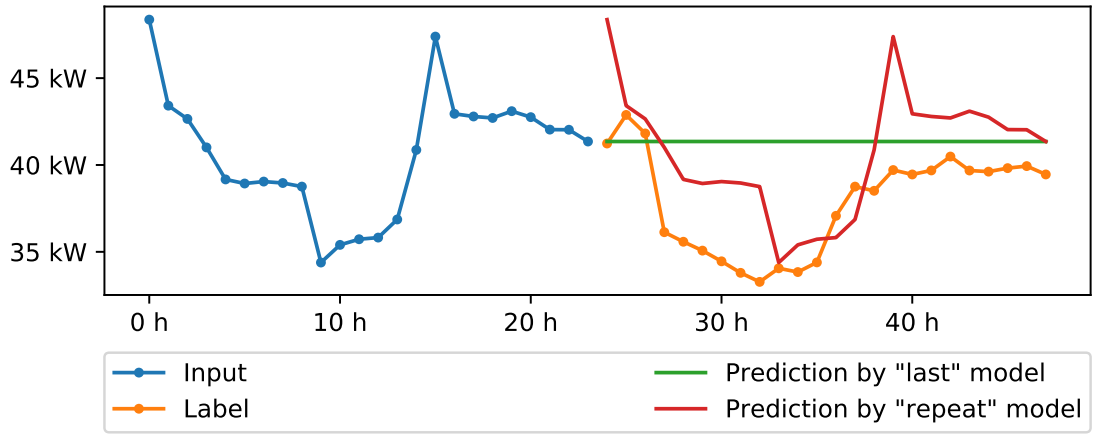

Figure 2.1: Example of the two baselines predicting the next day of power consumption data based on the previous one. This example is taken from the test dataset.

2.4 Artificial Neural Networks

Statistical methods for time series prediction are limited in their ability to work with complex and non-linear systems. Artificial neural networks can overcome this by using multiple layers of neurons in order to produce complex outputs. Neurons in their simplest form are nodes in a graph that put values through an activation function and then pass them on to be used by other neurons further down the line. The biases of the neurons and the weights of the edges connecting them are trained using backpropagation (see [16] for details on this process). Given the desired output of a network for a certain input, the optimizer updates the weights and biases, also called *parameters*, of the model $\mathbf{F}$ such that it will produce an output with a smaller loss $E_{\mathbf{F}}$, the value of the *loss function*, which is the optimization target. An *epoch* of training is completed by training the model with the entire training dataset once. Usually multiple epochs are necessary to fully train a model. Learning neural networks with multiple layers is referred to as *deep learning*.

These models make an entire prediction consisting of m steps without interruption instead of successively predicting single steps as it was defined in the problem definition. These *multi-step* models can be thought of as a group of individual models $\mathbf{F}_i$ that each produce the single prediction vector $\hat{\vec{y}}(t+d)$ with lag $d = i \in [1, m-1]$ based on their shared input. They are effectively just an efficient combination of many *single-step* models.

Dense Models

In *dense* layers each neuron is connected to each other neuron. They can model linear relationships and more complex ones when combined into a network of multiple such layers. Dense models make use of one or more dense layers. A model with only one input and one output layer can model linear relationships using the connections between the two layers. Here such a model is referred to as having zero dense layers even though there are trainable parameters in the model. Figure 2.2 is a diagram of a dense model.

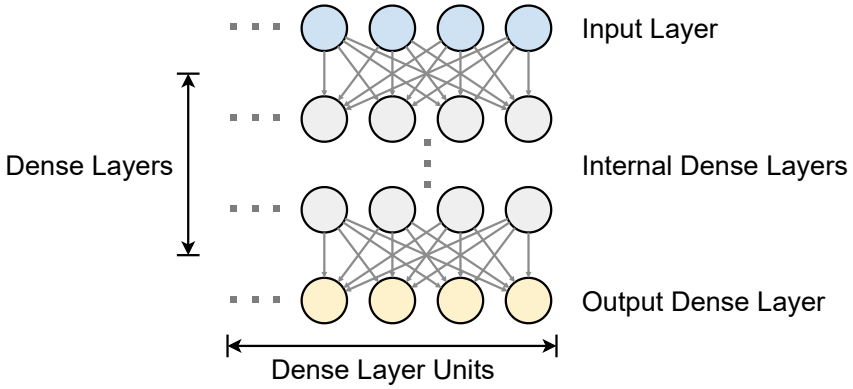

Figure 2.2: This diagram represents a dense model as a graph of nodes and edges. In practice, model parameters are a stored and operated on as matrices. The dense layers extend further to the left.

Convolution Models

Convolution layers in neural networks contain a convolution kernel. Instead of making every possible connection between input and outputs, the kernel slides over the input and produces an output using a section of the input weighted by the kernel. This means the weights are to a degree shared between all sections of the input. A kernel can be trained to recognize patterns without needing patterns to appear at all possible positions in the training data as it would be necessary for a regular dense layer. This beneficial effect is called *parameter sharing* and is based on the assumption that learned patterns from one part of the input can be applied to other parts as well. More information about convolutional neural networks and their use in image processing is given by O'Shea and Nash [30]. Figure 2.3 on the following page is a diagram of a convolution model.

Long Short-Term Memory Models

LSTM was introduced by Hochreiter and Schmidhuber and outperformed other recurrent neural networks of the time on tasks involving time series data [17]. As the name Long Short-Term Memory suggests, LSTM units remember things over time by maintaining an internal state throughout the time series. Gates within the units regulate the flow of information in and out of them. A detailed and mathematical description of how LSTM works can be found in the original paper by Hochreiter and Schmidhuber [17].

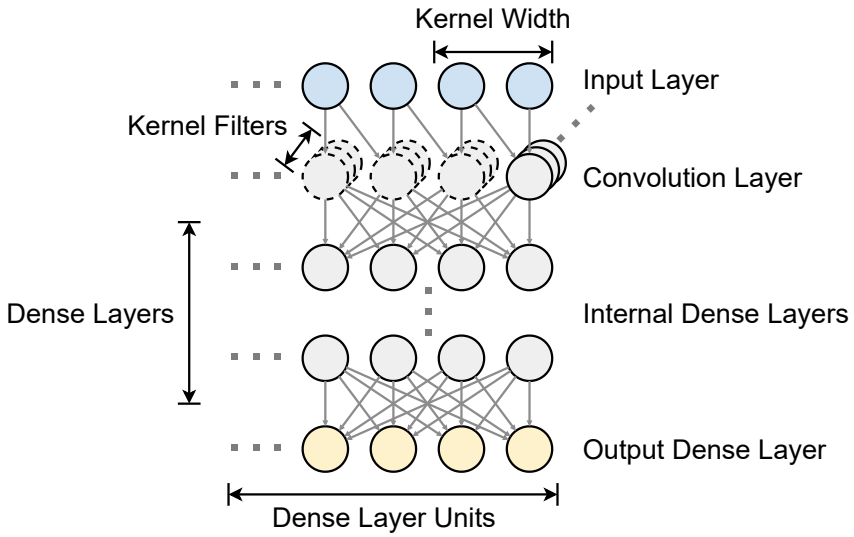

Figure 2.3: This diagram represents a convolution model as a graph of nodes and edges. The nodes with dashed outlines are not separate nodes but rather positions in which the nodes perform calculations.

Autoregressive LSTM Models

An autoregressive (AR) LSTM neural network works in two phases, a warmup and an output phase. In the warmup phase, the LSTM cell reads the inputs and prepares the internal state. In the output phase, the output of the cell at each iteration is used as the input for the next while also being saved as the output of the model. This architecture allows the output length to be varied after training. It is considered autoregressive since it uses its own output as the input for the following step. Figure 2.4 illustrates the structure of this model type.

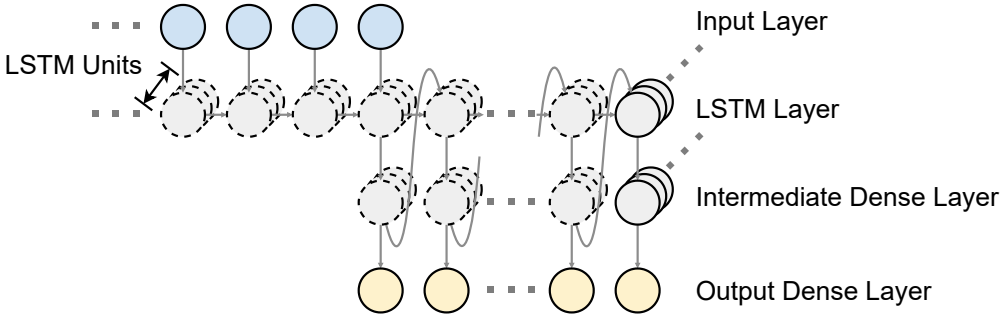

Figure 2.4: This diagram represents an AR LSTM model as a graph of nodes and edges. The LSTM cell's internal state is passed horizontally along the chain.

3

Application Context

The increasing demand for renewable energy sources requires a solution for the fluctuating power generation it entails. One solution is battery-based grid stabilization, which has a fast response time and is cost-effective for balancing energy generation and consumption over short periods of time. Batteries can be charged and discharged depending on the current grid requirements and can rapidly respond to load peaks or dips. Tesla, Inc. has demonstrated that high-power short-term frequency stabilization of power grids using large batteries is possible with their construction of the Hornsdale Power Reserve in Australia [31]. Because there is an increasing number of electric vehicles attached to charging stations at any one point in time throughout the grid, the batteries in electric vehicles can be used as a distributed battery farm for grid stabilization. Compensating for loads on the grid using vehicle batteries and dynamic charging schedules is known as a *vehicle-to-grid* (V2G) system. Another mode of operation is unidirectional V2G, called *V1G*, where charging power is temporarily reduced in response to grid requirements. *VxG* is a collective term referring to general bidirectional charging and load balancing as it combines V1G and V2G.

3.1 ReNuBiL

The ReNuBiL project implements VxG for two shared electric vehicles and charging stations on the campus of the University of Lübeck. As part of that effort, methods are evaluated that make bidirectional charging and in particular returning power to the grid more efficient. The goal is to charge electric vehicles when renewable energy is abundant, the electricity price is low and during times at which no peak loads need to be compensated. Additionally, it is desirable to compensate peak loads whenever possible. Honoring all fulfillable user bookings, however, stands above these goals.

A scheduling process will try to ensure that the vehicles are attached to the charging stations as much as possible during times when interacting with the grid would be most beneficial. The calculated schedule is realized by making suggestions to users in order to influence their bookings and by controlling the bidirectional charging stations. It uses a prediction of the power consumption of the Audimax building as well as predictions of the renewable power generation volume and electricity prices to determine the

best times for returning or receiving power from the grid. Figure 3.1 shows a photo of ReNuBiL's electric vehicles at the bidirectional charging stations while figure 3.2 on the next page shows a screenshot of the Grafana dashboard, which displays the latest power consumption data from the meters.

Figure 3.1: A photo of the electric vehicles for ReNuBiL. Source: © Martin Sachenbacher, used with permission.

Knowing this data in advance, or at least a sufficient approximation of it, should improve scheduling outcomes on average in regards to the aforementioned goals. In this project, due to the limited scale of the lab there is no coordination with the grid provider who knows if there is actually a deficit or surplus of power in the network. While the power consumption of the Audimax may be an indicator of the situation in the grid, many other factors can influence whether or not returning power to the grid at any point in time is beneficial to the grid provider's or the building owner's goals. The electricity price and the renewable power generation can also be indicators of the grid situation but they too do not provide a full picture. However, this last consideration lies beyond the scope of the ReNuBiL project and this work.

3.2 Energy Data

Wholesale electricity price data was kindly provided by the Stadtwerke Lübeck GmbH, the local utility company. They have requested that raw values from this dataset are not shared outside the university. The dataset consists of hourly prices in €/MWh over a time span of just over three months.

The power consumption of the Audimax lecture building on the University of Lübeck's campus is measured by a meter attached to the building's power connection. Current, voltage and wattage measurements have been saved to a database every 10 seconds starting in January 2021. Since the charging station is attached to the Audimax's power sup-

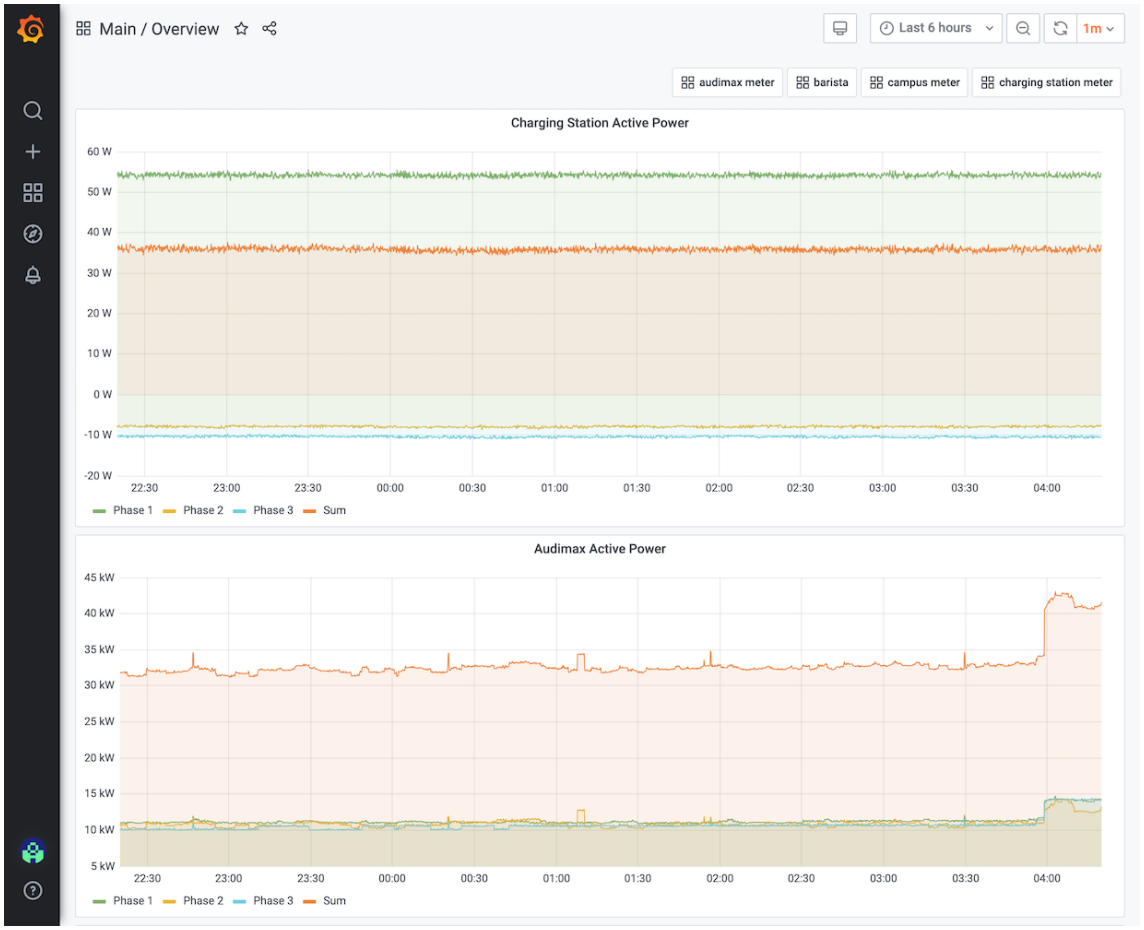

Figure 3.2: A screenshot of the Grafana dashboard, which displays the latest power consumption data from the meters.

ply, measurements by a different meter for the charger are subtracted from the building's power readings.

Power generation volume is used as a rough measure of how much renewable power is available on the grid. Data of the volume of renewable power generation is derived from the *Actual Generation by Production Type* category on the ENTSO-E Transparency Platform [13] for the years 2018 to 2020 and 2021 up until the 15th of August. This dataset contains the power generation values for all types of generation in 15 minute intervals. The sum of the current actual generation from all renewable production types is used as the main value for this data source. These production types include *Biomass*, *Geothermal*, *Hydro Run-of-river and poundage*, *Hydro Water Reservoir*, *Other renewable*, *Solar*, *Wind Offshore* and *Wind Onshore*. Nuclear and waste incineration power are excluded as they are generally not considered renewable energy sources.

3.3 Forecasts for Scheduling

The goal is to provide the scheduler in ReNuBiL with predictions of the three data sources that allow it to make better decisions. Additionally their accuracy should be high enough for the predictions to be useful. This makes it important to define the time scale and period of the problem. The forecasts are useful when the scheduler can better compensate peak loads and use abundant renewable energy more often with the forecasts than without them. Of course the magnitude of this improvement is also important. For scheduling vehicle bookings, a forecast of the energy trends over the next day is the most interesting. Predicting a longer time frame would not yield much better results than simply repeating the already predicted day because of the high complexity and noise within the systems. A shorter time frame would make scheduling harder as the durations of the events with which the scheduler works are each a few hours long. This includes vehicle bookings, vehicle charging and discharging batteries.

The best resolution for the predictions is around a half to four samples per hour depending on how accurately the predictions can be made and if the scheduler can make proper use of higher resolutions. The data has to be downsampled to reduce its resolution if it is too high since processing it with unnecessary precision would incur a performance and training time cost. The input data may be chosen to have a slightly higher temporal resolution than the prediction in case additional hints about future trends can be extracted from more detailed input data. For the selection of model types and their configurations in the following chapters, the output resolution is one sample per hour and forecasts are made up to one day into the future. If need be, these application parameters can be adjusted in response to new requirements.

3.4 Statistical Baselines

For this specific use case avoiding large individual errors is important as they can affect the scheduler's decisions. Therefore MSE is chosen as the loss function for training models and as the objective metric for hyperparameter tuning (see section 5.3 on page 24). However, the MAE is also measured during tuning and training for a different perspective. Results with MAE values are included in the raw result data but not in this document directly.

Statistical methods, such as linear regression, could be applied to the problem but they are unlikely to be successful because of the data's structure. The time series are not simple trends but rather contain many complex patterns. The power consumption data, for example, features sharp changes around specific times of each day, a lower power consumption at night and other varying patterns, all of which are probably created by the building's climate control systems and human activity. Nevertheless, simple statistical methods are used as baselines against which more complex models are benchmarked.

4

Data Sources and Preprocessing

4.1 Primary Data Sources

The electricity prices, power consumption and power generation are considered the primary data sources. Important details about each data source are described in this section while they are initially introduced in section 3.2 on page 11.

Electricity Prices

Due to the relatively small time frame of the electricity price dataset, model performance might be reduced because of a lack of training data. Since the data only covers the time from January 1 to April 7, 2020, a model trained on this data cannot observe possible long-term seasonal patterns.

Audimax Power Consumption

There are sometimes gaps in the power consumption data when some part of the system is interrupted. These gaps in both streams of data are filled in with the last known value. Additionally, the two meters' measurement intervals do not perfectly align nor are they perfectly spaced. To solve this, both time series are re-indexed onto a perfectly aligned 10 second measurement interval using linear interpolation. Then the correction of the building's own power consumption with the charger's consumption can be performed. While this method is not perfect, it results in relatively accurate power consumption measurements for the building. Table 4.1 on the following page lists the number of gaps and total time for which samples are missing in the data for both components. The long gaps are generally caused by unscheduled downtime of the measurement and recording systems while the short gaps are likely the result of temporary interruptions or system restarts. Preprocessing of the data reduces the temporal resolution enough for small errors at the level of individual measurements to be smoothed out.

This dataset is continuously expanding as measurements are still being taken. The starting date January 14, 2021 and the cutoff date August 26, 2021 yield around eight months of data of which only seven are usable since there are large measurement errors

at the beginning. In order for all experiments to be based on the same data, the cutoff date was set just before the first experiments that are presented in this work were started.

Component	Long gaps	Missing time from long gaps	Short gaps	Missing time from short gaps
Audimax	25	40h 28m	317	0h 44m
Charging station	35	5h 19m	966	3h 27m

Table 4.1: The number of long (over one minute) and short (between 10.5 seconds and one minute) gaps in the data for both components together with the total length of time for which measurements are missing as a result of these gaps. For example, in a 15 second gap only five seconds would be counted as missing since the first ten seconds are still covered by the last recorded measurement.

Renewable Power Generation

While the ENTSO-E Transparency Platform has renewable power generation data spanning many years, the section of that data used in this work is around three and a half years long starting on January 1, 2018 and ending on August 15, 2021. If a row is missing a value for any production type, it is treated as invalid and the corresponding sum replaces the last known value.

4.2 Time Signal Inputs

Besides the data values that are used as a basis for the prediction, six time signals are passed into the model so that it has access to information on the time of day, the day of the week and the day of the year. A sine and cosine of the fraction of the current day, week and year are added as input columns. This allows the model to better reproduce possible repeating patterns by training it to incorporate possible correlations between these time signals and the data values into its predictions. Zhang and Kline found this approach using trig functions to be beneficial to model performance [47].

4.3 Air Temperature as an Additional Input

While the building's heating and cooling power consumption is determined by a complicated interaction of outside temperatures, heat from the sun and precipitation amongst other factors, the outside air temperature could be a rough indicator of how much energy might be required for climate control. Data from hourly measurements of the air temperature two meters above ground near the Airport Blankensee, which is located around three kilometers from the Audimax, is kindly provided by the German Weather Service (Deutscher Wetterdienst, DWD) [11]. Figure 4.2 on the next page shows examples of time signals and air temperature inputs.

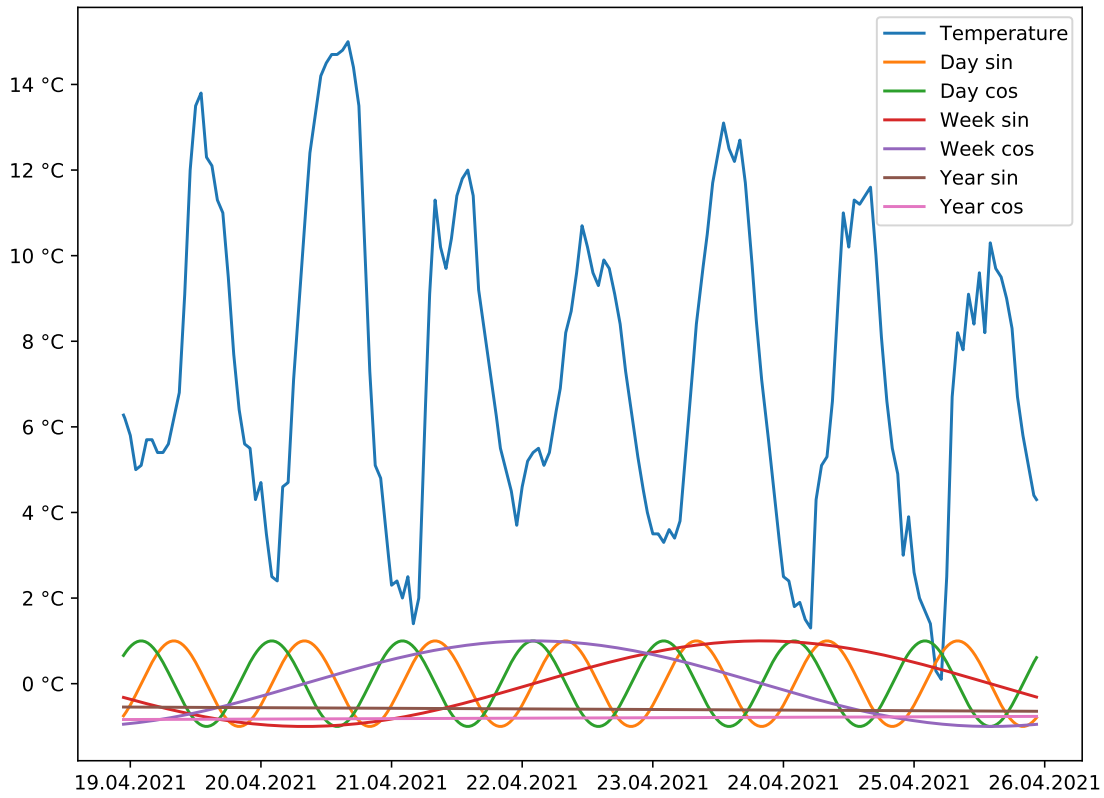

Figure 4.2: Example of the time signal and air temperature inputs over the span of one week. All plots share the same axis but the time signals are unitless.

4.4 Downsampling

The power consumption data is recorded with a high sampling frequency, which results in a large amount of samples. The downloaded data for the Audimax and charging station meters has a combined file size of 383 megabytes. Processing this data and training models on it without reducing the resolution is unwieldy and slow. Therefore, it is downsampled to use a longer sampling interval of one hour. The length of the input data is important to consider in particular when training LSTM models because they become harder and much slower to train with very long inputs. In figure 4.3 on page 20, examples of the power consumption data are shown. The price data is not downsampled since its temporal resolution is already one hour. Furthermore the difference in resolution before and after downsampling is hardly visible for the power generation data because of its smoothness.

The downsampling process has the additional desirable effect of resulting in perfectly aligned samples. This is only the case because samples at the required times are newly calculated instead of being produced by simply omitting all unneeded samples. Samples taken at slightly random intervals interpreted as aligned intervals would distort parts of the data where there are more or less samples than expected.

4.5 Normalization

All input columns are individually normalized, here through standardization specifically, by subtracting the mean μ and dividing by the standard deviation σ of each column. In formal statistical terms the operation is the following where X is a random variable representing the elements of a time series:

$$X' = \frac{X - \mu}{\sigma}$$

Model performance is improved through the removal of scale differences between the inputs, thus eliminating the need for the model to handle scaling itself. Additionally normalization of inputs also avoids saturating activation functions within neural networks [19]. Removing an offset and linear scaling from the data reduces the amount of unnecessary work the model has to do. This process of normalization is reversed on the output data to obtain the values in their original ranges:

$$X = X' \cdot \sigma + \mu$$

Inputs are normalized before being passed to a model and denormalized for use outside the model. However, error metrics are calculated on the normalized model outputs since this allows errors to be roughly compared between data sources and avoids dealing with large values.

4.6 Seasonal Adjustment with Median Windowing

A time series $\vec{x}$ can be modeled as being composed of noise basis $\vec{b}$ and seasonal component $\vec{s}$ contributed to by a number of prominent seasonalities $\vec{s}_i$ with periods p_i where $i \in \mathbb{N}$ is in practice 1 or 2:

$$\vec{x} = \vec{b} + \vec{s}_1 + \vec{s}_2 + \ldots$$

There is some debate on whether seasonal adjustment, which removes seasonally repeating patterns from the data before feeding it into predictive models, is helpful or not. In their experiments Zhang and Kline find that it is helpful [47]. They also found simpler models to outperform more complex ones in general. In this work seasonal adjustment is evaluated as a way of improving model performance alongside other methods. Daily or even weekly repeating patterns are clearly visible in all three data sources. Seasonal adjustment attempts to remove these seasonalities $\vec{s}$ leaving the model to work with the noise basis $\vec{b}$ of the data. Despite the name, seasonalities can have a period p of any number of samples and do not need to be seasonal in the traditional sense of the word.

The most straightforward way of adjusting for daily seasonalities is to calculate the median over the whole dataset at each sampling time of the day. Given a sampling frequency of one sample per hour there are $p = 24$ samples in a day. This calculation yields a median day $s_{\text{md}}(t_d)$ that is a function of the sample index $t_d \in [0, p-1]$ within the

seasonality's period p. t_d corresponds to the time of day if the period p is one day worth of samples long.

$$\vec{s}_{\text{md}}(t_d) = \text{median}\{\, \vec{x}(p \cdot d + t_d) \mid d \in \mathbb{N}_0 \,\}$$

For this method the seasonally adjusted data is calculated as

$$\vec{x}_{\text{md}}(t) = \vec{x}(t) - \vec{s}_{\text{md}}(t \bmod p).$$

In these datasets the daily seasonalities remain relatively stable only over short periods of time and change over the course of weeks. In formal terms, this means $\vec{s}_{\text{md}}$ is not always a good approximation of the seasonal component $\vec{s}$ in $\vec{x}$. This drift can make the seasonally adjusted data $\vec{x}_{\text{md}}$ more complex for a model to predict than the original data $\vec{x}$ itself, rendering this simple approach ineffective.

A solution to this problem is to instead use a sliding window to calculate the local seasonality for each day. The window is w seasonality periods long. The median over the values of the last $w = 7$ days at the same time of day is calculated as the local seasonality $\vec{s}_{\text{mw}}(t)$. For example, the local seasonality for a data point at 13:15 is calculated from the median of the values at 13:15 on the previous seven days. The local daily seasonality

$$\vec{s}_{\text{mw}}(t) = \text{median}\{\, \vec{x}(t - p \cdot d) \mid d \in [0, w - 1] \,\}$$

can be used to produce the corresponding seasonally adjusted time series

$$\vec{x}_{\text{mw}}(t) = \vec{x}(t) - \vec{s}_{\text{mw}}(t).$$

The optimal window length and method are not immediately obvious from the data. Window lengths such as a single day, multiple weeks or some other number of days could be effective. The average magnitude of the seasonally adjusted data serves as a proxy for the effectiveness of a given windowing period. This is an imperfect measure but it does provide some interesting insights into the patterns that can be found in the data. The rationale is that a more thorough seasonal adjustment can better disburden the model of predicting seasonalities. The results of the following window length analysis are similar to a frequency decomposition.

Figure 4.4 on page 21 compares the window lengths w and the windowing methods mean and median. For the price data a window length of around two months has the smallest difference magnitude with the mean window but choosing such a long window would severely reduce the amount of available data since the price dataset is only 97 days long. With this in mind the best windowing method is taking the median and the best window length is one week for both the price and power consumption data. The dips in the difference magnitudes at multiples of seven days in both the price and power consumption data are not surprising, given the weekly cycle these systems have. In both cases different power consumption behavior on weekends and to a lesser extent on each weekday can cause aligning the window length to full weeks produce particularly low difference magnitudes.

The best window length for the power generation data is one day. Because renewable power generation is mostly influenced by the weather, a weekly seasonality is not to be

expected, even though the grid operator may slightly adjust generation to match demand. The small reduction in the difference magnitude around a window length of 40 days may be an effect of monthly weather variation. Even longer window lengths could be producing larger differences because they include data across multiple seasons.

Figure 4.5 on page 22 shows examples from all three data sources together with the seasonally adjusted data and the corresponding seasonalities.

4.7 Qualitative Prediction

The consumers of the predictions actually only need quite coarse information about the probable behavior of the three data sources over the course of the next day. Even predictions with a resolution of one hour could be more frequent than they need to be. However, reducing the output resolution alone just reduces the amount of information produced by the models and provides few benefits beyond a slightly reduced training time. Having the models predict additional qualitative information about the data could give the consumers information that is more helpful than if simply more data points were provided. Such additional outputs could include the spread, minimum, maximum and standard deviation over the interval each output data point represents. These outputs are calculated using the slightly higher resolution version of the data, which the model does not need to directly predict. How valuable this qualitative prediction is compared to the regular single-output version depends on what the predictions are used for. Model performance using qualitative prediction is evaluated in the Method Evaluation section 6.5 on page 35.

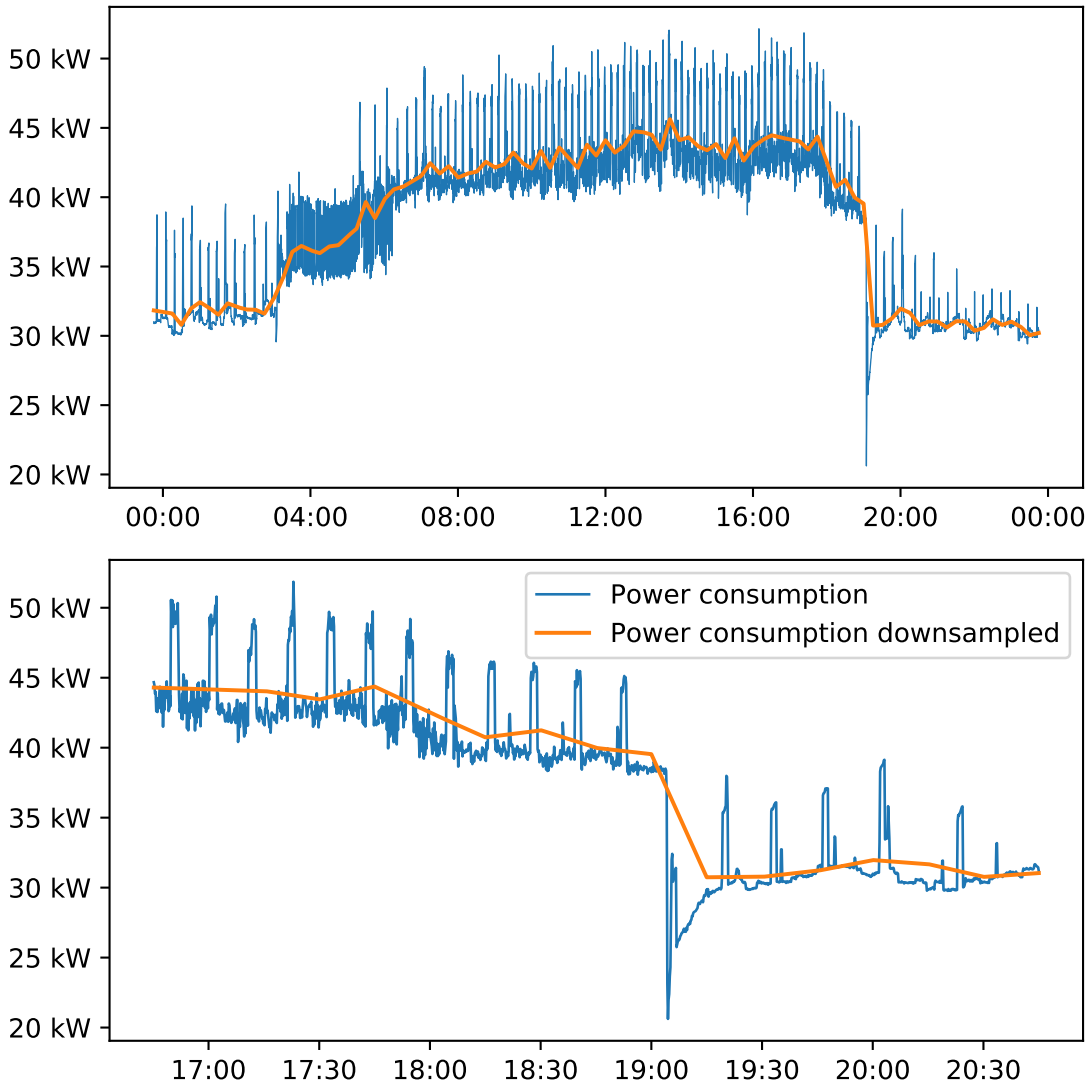

Figure 4.3: An excerpt of the power consumption data in unprocessed and in downsampled form.

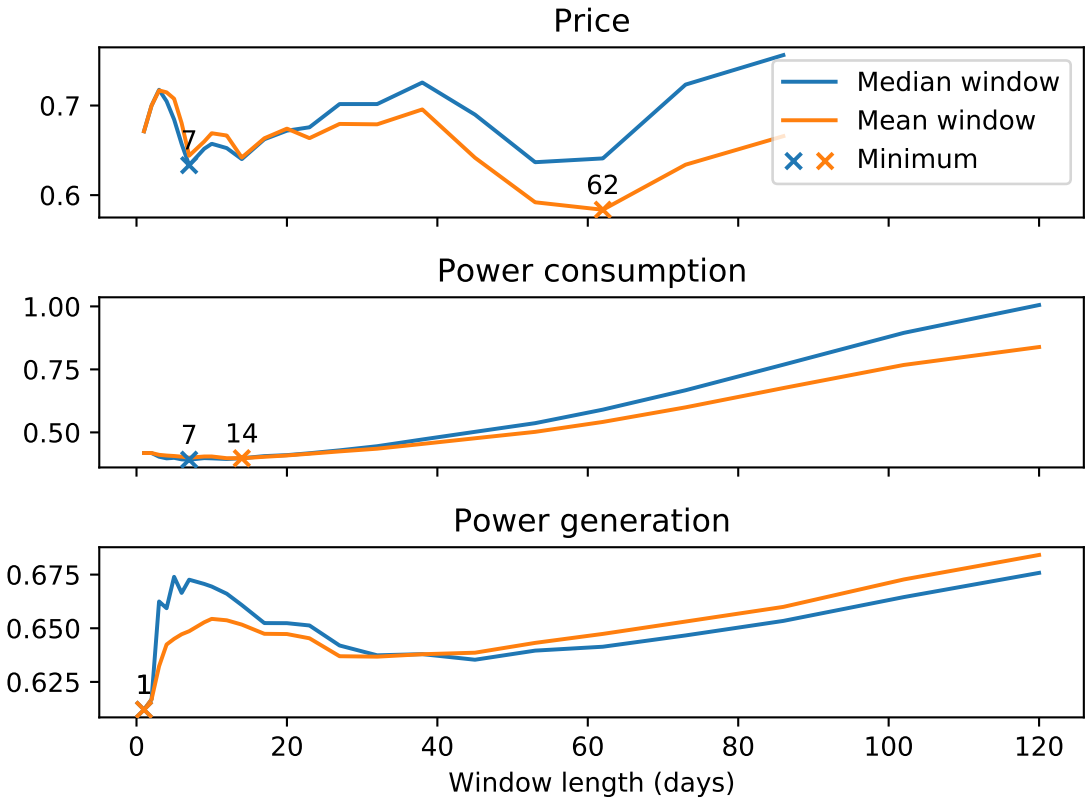

Figure 4.4: A comparison of the different window lengths and methods. The crosses mark the smallest value for each windowing method and are annotated with the corresponding window length w. In the bottom plot the crosses are at exactly the same position causing only one to be visible.

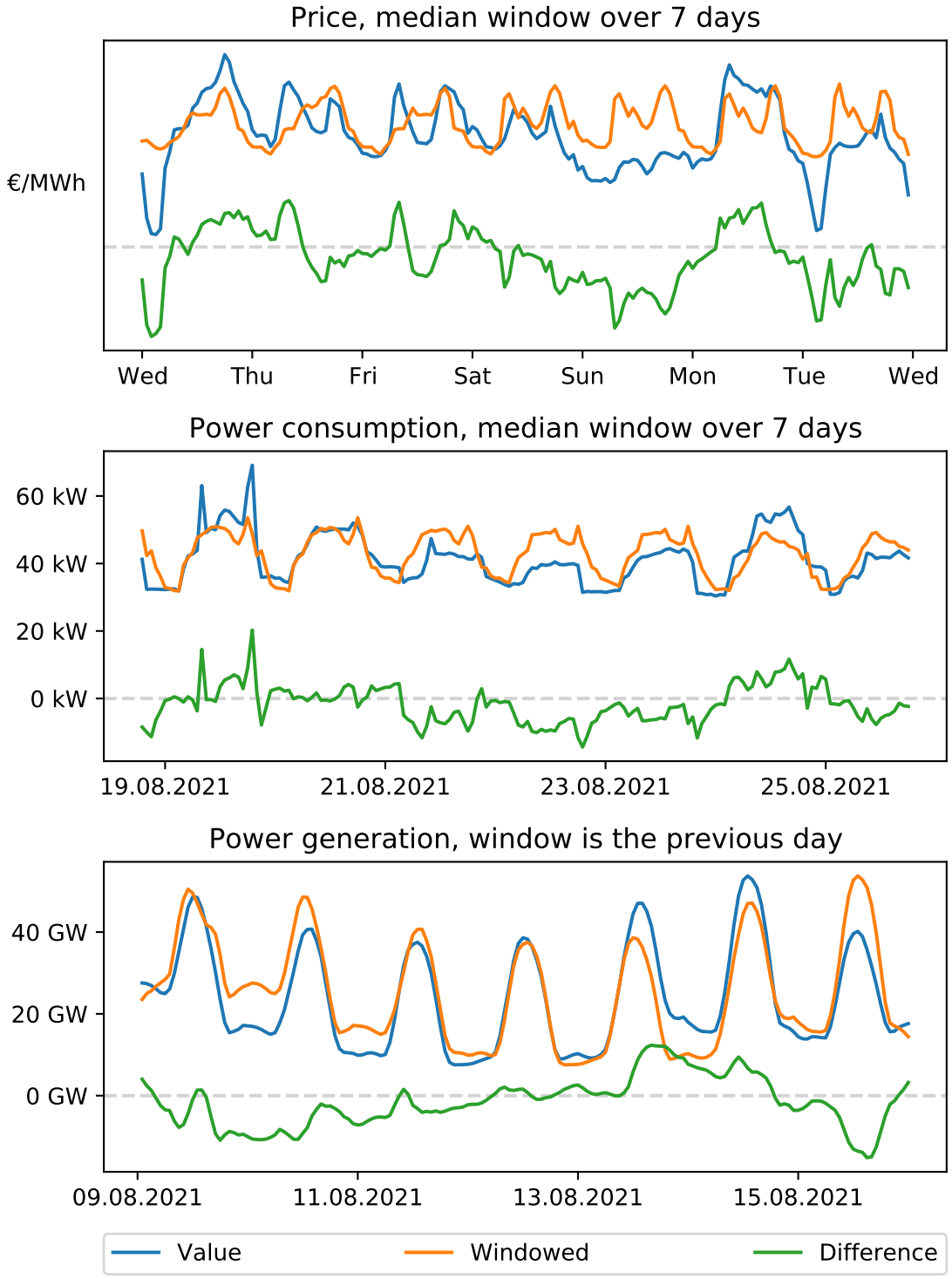

Figure 4.5: One-week-long excerpts of each data source before and after subtracting the seasonalities obtained with the sliding median window. The subtracted daily seasonality is also included. Notably, after seasonal adjustment the data has a smaller magnitude and is less cyclic.

5

Training Preparation and Hyperparameter Tuning

5.1 Dataset Splitting

After training a model on a dataset, the model is very good at recognizing pieces of its training data and giving good predictions on it. Though this does not necessarily mean the model can make useful predictions on unknown data since it could have also simply memorized the training data instead of generalizing from it. In order to test the performance of a trained model it needs to be tested with previously unseen data. In particular, it is important that the model's training gives it no indication of the exact details of the testing data. If a model has direct knowledge of the testing data then a test using that data would be biased in the model's favor. A model is considered to be overfit when it performs well on training data but badly on unknown data. An under-fit model on the other hand needs more training and does not perform well yet on any data.

A Three-Way Split

Evaluating model performance requires a test dataset. However, using the same dataset for model tuning and for the final evaluation introduces the risk of overfitting on the testing data. Choosing a model out of a potentially large pool of models based on testing data can lead to the selection of a model that happens to perform well on the specific data chosen as the test dataset but badly on real-world data. Therefore, another set of testing data that is only used for the final evaluation is necessary. This leads to a three-way split of the prepared dataset: 80% of the data is for training, 10% of the data is for validation and the remaining 10% are for testing. The search for the best model is performed on the validation data while the final evaluation verifies the choice using the test data. No adjustments to the models can be made based on the result of the final evaluation or a bias could be introduced. The validation and test datasets should be representative of the training dataset but they have to all remain strictly separate.

5.2 Baseline Performance

If a neural network-based model performs better than the baseline on average then it is considered useful by this benchmark and should be considered for use in the real-world application. The error values produced by the two statistical baselines $\mathbf{F}_{\text{last}}$ and $\mathbf{F}_{\text{repeat}}$ for each data source can be found in table 5.1. These error values are calculated on the normalized values and are, therefore, unitless.

Baseline	Dataset	MSE Price	MSE Consumption	MSE Generation
$\mathbf{F}_{\text{last}}$	Validation	0.828	1.212	0.774
	Test	0.826	1.676	1.649
$\mathbf{F}_{\text{repeat}}$	Validation	0.744	0.677	0.930
	Test	0.735	0.542	0.477

Table 5.1: The MSE of the two simple baselines with the validation and test datasets.

No Input Baseline

It is interesting to quantify a model's dependence on the given inputs for making predictions. A model that is given no inputs containing data from a primary data source, such as power consumption data for example, can only rely on memorizing the training labels, which are the expected outputs for a given set of training inputs. A model's capacity for memorizing outputs during training is evaluated with the *no input baseline*. This baseline can be constructed from any model by training it with all of its primary data inputs zeroed out. The no input baseline can only use the additional data sources, namely time signals and temperature data, to make predictions. A model performing better than this baseline is likely using the given inputs to produce an output and is not making predictions solely based on memorized training data.

A vector $\vec{x}(t) = (p_1, \dots, p_n, a_1, \dots, a_m)$ in a time series $\vec{x}$ contains the input values for one time step t. Of these values, $p_1, \dots, p_n$ are from a primary data source such as the electricity price, power consumption and power generation. The values $a_1, \dots, a_m$ on the other hand, are additional inputs such as time signals and temperature data. A regular model $\mathbf{F}$ receives a sequence of vectors from $\vec{x}$ to make a prediction of $\vec{y}$. The no input baseline $\mathbf{F}'$ receives partial input vectors of the form $(0, \dots, 0, a_1, \dots, a_m)$, which are derived from those in $\vec{x}$, but is trained to predict $\vec{y}$ like a regular model. The resulting model, to a degree, captures the correlation between the additional inputs $a_1, \dots, a_m$ and the expected output $\vec{y}$. This baseline is different from the statistical baselines as they do use primary data inputs while this one does not.

5.3 Hyperparameter Tuning

Hyperparameter tuning is the manual or automated optimization of a model's hyperparameters, which can have a large effect on performance by indirect means. They include

fixed model parameters, such as the type, number and size of neural network layers and parameters of the optimizer, such as the *learning rate* (LR). They are called *hyperparameters* because they are parameters that are fixed before training whereas the actual parameters of the model, such as the layer weights and biases, directly determine the model's output but are calculated during training. The performance of a model with a given set of hyperparameters is impossible to determine precisely without actually training and evaluating it.

Instead of testing all possible combinations of hyperparameter values, different methods of selecting hyperparameters may be utilized. The three offered in KerasTuner [29] are a random search of the hyperparameter space, Bayesian optimization [39] and Hyperband [24]. In this work the Bayesian optimization tuning algorithm is used. In contrast to Hyperband, Bayesian optimization trains each model completely before moving on to the next iteration. Some of the models used in this work have been observed not to have steadily increasing losses during training, which can disrupt Hyperband's search strategy.

KerasTuner allows each model to be trained multiple times for each set of hyperparameters in order to increase the accuracy with which the performance of the hyperparameters is evaluated. This leads to a more stable search result, which is important considering the substantial variation in performance that individual trained models display. The tuner tests 50 different sets of hyperparameters for each of which three model instances are trained. This means the Bayesian optimization algorithm chooses a random set of hyperparameters for the first iteration and then chooses new sets of hyperparameters for each following iteration based on all the previous iterations' results.

Hyperparameters

Each model has its own set of hyperparameters depending on the type and structure of the model. They are defined within a function that constructs a model instance. The common hyperparameter is the learning rate parameter for the optimizer, while all others vary for each model. Table 5.2 on the next page gives an overview of the model specific hyperparameters. Figures 2.2 to 2.4 on pages 8–9 show the hyperparameters within diagrams of each model's structure and what parts of the models they affect.

In the dense model, the number of dense layers and each of those layer's sizes, also referred to as *units*, are hyperparameters. Conditional hyperparameters are used for the layer units since they are each only applicable if a certain number of layers has been selected.

The convolution model has a number of tunable hyperparameters. The convolution layer has the kernel width, which is the number of inputs the kernel is applied to at a time, and the number of kernel filters, which are separate sets of kernel weights. This means both dimensions of the kernel parameter matrix can be tuned. Another hyperparameter determines how many of up to two additional dense layers are added following the convolution layer. Each of these layers' sizes can be tuned as well.

The AR LSTM model's only hyperparameter is the number of units in the LSTM cell.

Model	Hyperparameter	Value Interval
Dense	Extra Dense Layers	$[0, 3]$
	Dense Layer Units (each)	$[16, 512]$
Convolution	Kernel Width	$[2, 12]$
	Kernel Filters	$[8, 128]$
	Extra Dense Layers	$[0, 2]$
	Dense Layer Units (each)	$[16, 512]$
AR LSTM	LSTM Units	$[256, 512]$

Table 5.2: An overview of the hyperparameters that can be tuned for each of the three models.

6

Evaluation

6.1 Implementation and Model Training

Here a few technical details are discussed in order to give a rough impression of the implementation. The repository containing the source code, the raw data and the evaluation results can be found in the accompanying materials[1]. In the Python script the data is loaded from CSV (comma separated value) files using Pandas [41, 27] into `DataFrame` objects. A data frame is a two dimensional table with columns and rows. Each row is one data point, essentially a vector, that has values for each of the columns. The models all have only one input, namely the main data frame. The columns of the data frame are also referred to as multiple inputs or input columns. The same goes for multiple outputs, which are here simply multiple output columns. Models with multiple separate inputs and outputs are possible in general but not used in this work.

An interesting implementation detail concerning the power consumption data is that it needs to be downloaded piecewise from the Grafana dashboard. Due to timeouts limiting the maximum duration of a query, the full dataset cannot be downloaded at once. Instead, it is manually downloaded in chunks and subsequently joined using a utility script. This process also analyses the gaps in the data. Large gaps with unusually regular lengths and positions can indicate missing chunks of data.

Preprocessing behavior and the selection of a data source is simplified through the use of `DataSource` classes for the data sources. A parent class contains methods for general processing using the settings and data loading methods defined in each individual data source classes. The Python script adapts and incorporates parts of the code from the TensorFlow time series forecasting tutorial [42].

TensorFlow and Keras

TensorFlow [1] with the Keras API [10] facilitates defining and working with many types of ANNs. TensorFlow provides the low-level computation services that Keras uses to present a high-level API for machine learning tasks. Models are trained using the Adam optimizer, which has been found to perform well on many tasks, and the same goes for

[1] The repository is also internally available at `https://gitlab.isp.uni-luebeck.de/renubil/renubil-timeseries-prediction`. It includes a digital copy of this text.

time series prediction tasks in particular. This makes it the recommended optimizer chosen by TensorFlow and the one used for training models in this work. Since Adam is a stochastic optimization method, training a model is a stochastic process and will yield randomly different results every time it is run. The evaluation of the effects of a choice on model performance using slightly random training outcomes is dealt with by training the multiple instances of the same model (see the Evaluation Methodology section 6.2 on page 30).

Keras allows for any number of error metrics to be recorded during training and evaluation. This is taken advantage of by specifying the MSE as the loss function and registering the MAE as an additional metric that Keras should report. In the Model Evaluation section 6.4 on page 32 only the MSE is given because of space constraints but the raw results do contain measurements of the MAE.

Synthesis of a Training Dataset

The dataset is split into training, validation and testing parts as discussed in section 5.1. Given a continuous data frame, which is effectively a table, discrete sequences for training the models have to be generated. In order to make efficient use of the data, the sequences on the dataset overlap except for an offset of one sample. This means each sequence is almost identical to the last but advanced by one sample. Another effect of these overlapping sequences is that the models are trained to predict at any position in the dataset and not only aligned to full days. This means the models are able to make predictions using any input data of the right length and format.

Each sequence consists of two parts, the input and the label values. Which columns appear in each part varies with the data source and which preprocessing techniques are being used. During training the input values are passed through the model and then the model's parameters are adjusted such that it better fits the expected labels. How many samples they contain is relevant internally but irrelevant in determining the fundamental behavior of the model. As mentioned previously, model inputs are one day long and they predict one day into the future starting immediately at the end of the input.

The generation of the dataset that TensorFlow uses to interface with the models is handled by the `WindowGenerator` class and specifically its subclasses. It uses a TensorFlow method to efficiently generate a time series dataset of sequences and randomizes their order to avoid a temporal bias during training. This is also the point at which input and label columns are separated into input and label sequences. Taking random samples after generating sequences with the window generator is not a good approach as it would lead to a substantial overlap between the datasets. For the same reason, the sequences are generated after splitting and not beforehand.

Hardware

TensorFlow trains the models using CPU or GPU cores and can optimize the calculations using hardware specific libraries like cuDNN [9] for NVIDIA devices. For this work the AI Lab Lübeck [3] of the University of Lübeck was used for the final training and evaluation of the models. It provides facilities for CPU and GPU training through a JupyterLab

environment where code is run in notebooks. One hardware option are Dell servers, which perform reasonably well for both hardware types. The other option is a NVIDIA DGX2 machine with GPUs that are faster than the Dell server's GPUs. In all hardware configurations, namely CPU on a regular desktop computer, CPU on the Dell server, GPU on the Dell server and on DGX2, TensorFlow automatically detected and optimized for the selected compute device with very little additional code being necessary.

There were some difficulties using the AI Lab and the DGX2 GPUs specifically. At first, GPU training did not work for LSTM models using cuDNN because dynamic allocation of GPU memory needed to be explicitly enabled in the code. Another issue was connecting to the notebook server and using the storage server. There was some unscheduled maintenance and there is a lack of documentation on the particulars of the Jupyter environment. Given that at the time of writing there is no proper scheduling system for using the DGX2 GPUs, scheduling is handled informally. How the selection of DGX2 GPUs works for the notebook and where this scheduling happens was not documented but became clear after detailed communication with the scientific staff familiar with the matter. Finally, DNS resolution failed during installation of Python packages in the notebook on the DGX2 GPU, which again made it impossible to run the notebook. The notebook was able to run on the DGX2 machine but the aforementioned problems repeatedly made it cumbersome to do so. The experiments for this work were performed using a GPU of one of the Dell servers.

Tuning and Training Times

The computation times largely depended on the data source and the size of the models, while the configuration had no impact. On the Dell server training a single epoch of the AR LSTM model took an average of five seconds for the electricity price data, ten seconds for the power consumption data and 70 seconds for the power generation data. For this model the mean best epoch varies between the first and the twelfth after which another fifteen epochs are trained to make sure the optimal performance was reached. While tuning took multiple hours for these first two data sources, it took over two days for the power generation data source. This disparity comes from the difference in available data, which proportionally affects how much data needs to be processed for each epoch. The computation of the seasonal adjustment and the addition of temperature data made no difference in training time. Interestingly, for the AR LSTM model even changing the number of LSTM units made no difference to the average training time per epoch.

Tuning and training the simpler models took significantly less time. For the dense model it took around one hour using the power consumption data, while it took around ten hours using the power generation data. The convolution model took a similar amount of time independent of the selected hyperparameters. On average, training the dense model for a single epoch took anywhere between 278 and 2725 milliseconds depending on the size of the input. The total training time also depends on the number of epochs the models are trained for, which varies between data sources and with the selected learning rate.

6.2 Evaluation Methodology

In order to find the best model for each prediction task, multiple levels of model configurations and parameters are optimized. First of all, the optimization is performed for each data source separately. Given a specific data source, the script searches for the best model. There are three *model types*: the dense model, the convolution model and the AR LSTM model. Each model can be configured in different ways. While not strictly hyperparameters, the choice of a model's inputs and outputs can have a large effect on its prediction performance. This includes seasonally adjusting the primary input data or adding temperature data. Any combination of these two options is referred to as a model *configuration*. With each option being either disabled or enabled, there are four possible configurations. One level deeper, model-specific hyperparameters and the learning rate are adjusted by hyperparameter tuning. More on this can be found in the Hyperparameter Tuning section 5.3 on page 24.

After hyperparameter tuning there is the option to train the model to predict the difference between its input values and the expected output values instead of predicting the output values themselves. In formal terms, this means a single-step model $\mathbf{F}$ is given a sequence of vectors $\vec{x}(t-i)$ with $i \in [0, n-1]$ and is trained to predict $\vec{y}(t+1) - \vec{x}(t)$. The vectors $\vec{x}$ and $\vec{y}$ are in this case required to have the same internal format and the dimensions k and h of the vectors have to be equal. This construction is also applicable to multi-step models but involves additional technicalities, which are omitted here.

With a given set of good hyperparameters, a model is trained in three different *modes*. The normal mode changes nothing. The second mode, referred to as the *delta* mode, trains the model on the difference between its input and the expected output as it was just defined. The third mode represents the no input baseline that is included for comparison. 20 instances of a model are trained in each of the three modes in order to acquire accurate performance metrics. To reduce training time, training is stopped once a model's validation loss has not improved for 15 epochs. The resulting metrics for each model instance are gathered separately for the training, validation and testing datasets and are finally combined by statistical methods like the mean, median and standard deviation. Since overfitting causes model performance to degrade after training for too many epochs, the metrics are recorded after the training epoch in which the model had the best validation loss.

In the delta mode and if the configuration includes seasonal adjustment, models are evaluated by adding their output to the seasonally adjusted data. Only in this situation are models trained to produce the seasonally adjusted time series $\vec{x}_{\mathrm{mw}}$ from which the seasonalities have been removed with median windowing. The process of seasonal adjustment is described in the Seasonal Adjustment with Median Windowing section 4.6 on page 17. In the normal mode the seasonally adjusted data, if enabled, is just an additional input.

For each data source, all combinations of model type, configuration and mode are compared using the metrics, and the combination with the lowest validation MSE is chosen as the best model. This result can then be validated, however not influenced, by the equivalent metrics obtained from the test dataset. Figure 6.1 on the following page is a simplified graphical representation of this evaluation methodology.

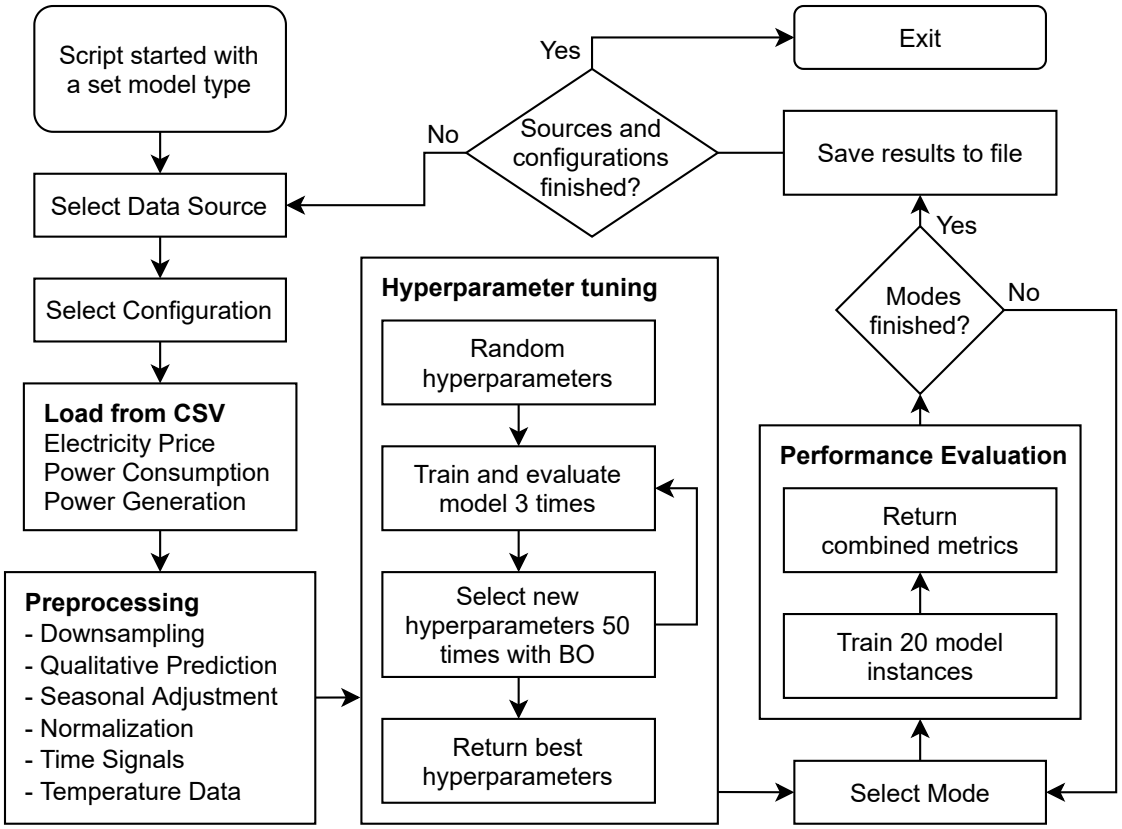

Figure 6.1: A flowchart of the evaluation methodology. BO stands for Bayesian optimization with which new hyperparameters are selected until all iterations are completed.

6.3 Tuning Results

The following sections discuss the tuned hyperparameter values for each model type. Figures 6.2 to 6.4 on pages 32–34 show the results of the three models predicting three segments from the power consumption test dataset. The prediction can be seen to sometimes deviate from the labels significantly in all three examples but it generally follows the course of the labels. Each trained instance of a model behaves slightly differently and its performance can vary depending on the specific input it is given.

The tuning results for the dense and convolution models vary greatly between configurations. One explanation for this large variation could be that the Bayesian optimizer was configured suboptimally. It is also possible that these hyperparameters may not have a large impact on the model performance and therefore were not tuned to be the same each time.

Tuning the dense models resulted in three- and zero-layer models for almost all configurations. The number of units in the three-layer models varies from 16 to 1024 without a discernible pattern. Together with the abundance of good zero-layer models this suggests that the model structure has a negligible effect on performance in this experiment. Additionally, the learning rate was tuned to the minimum value 0.0001 for all but two configurations.

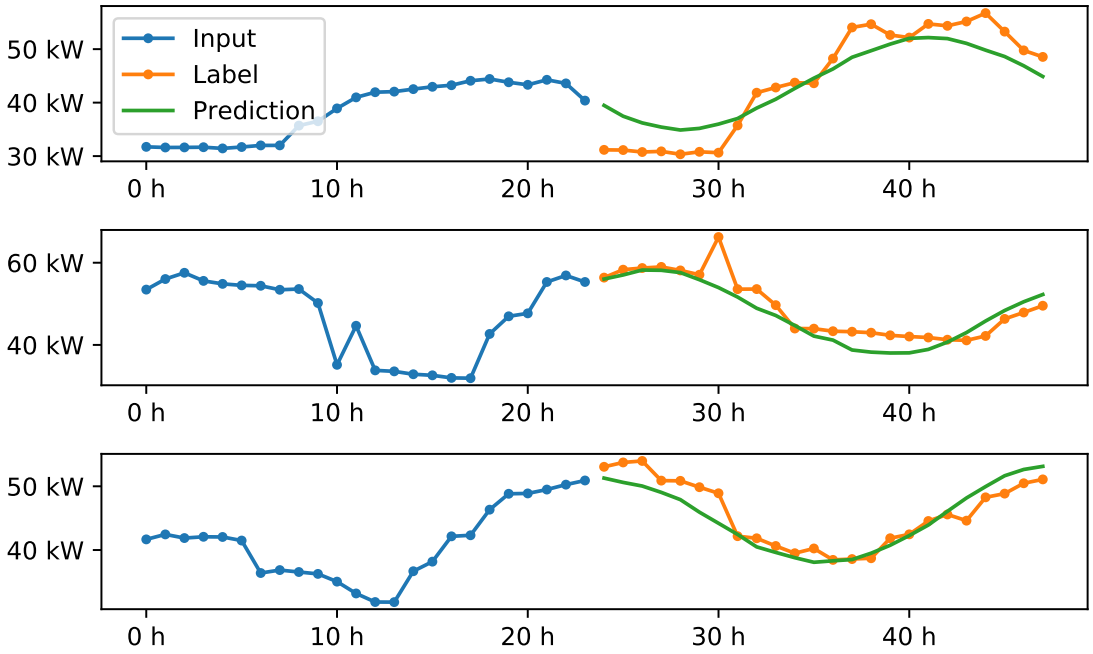

Figure 6.2: Three predictions on example days from the power consumption test dataset with a basic dense NN model with seasonal adjustment and temperature data. This model was trained using a learning rate of 0.0001 and 1024 units in a single dense layer, both of which were determined by the tuner for this model configuration.

The tuning results of the convolution models vary greatly between all tuning sessions with the only constant being the lack of additional dense layers for all but one configuration. This variation means that these hyperparameters only have a negligible impact on model performance. One explanation for the choice of no additional dense layers is the limited number of easily trainable patterns in the overall relatively small training dataset.

The number of LSTM units in the AR LSTM model was tuned to around 350 to 450 while the learning rate is between 0.0001 and 0.04. This tuning result is much more consistent across configurations compared to the other two model types. Because there is only one hyperparameter, it may be easier to tune accurately since finding its global optimum is likelier.

6.4 Model Evaluation

In tables 6.6 to 6.8 on pages 36–38 the results of tuning and training each of the three model types are presented. It should be noted that the error values can be somewhat skewed depending on the exact interactions of a model with the validation and testing datasets. The test results largely agree with the validation results and they have common best configurations for each data source. Additionally for almost every configuration, at least one of the modes is better than the no input baseline, which the test dataset also

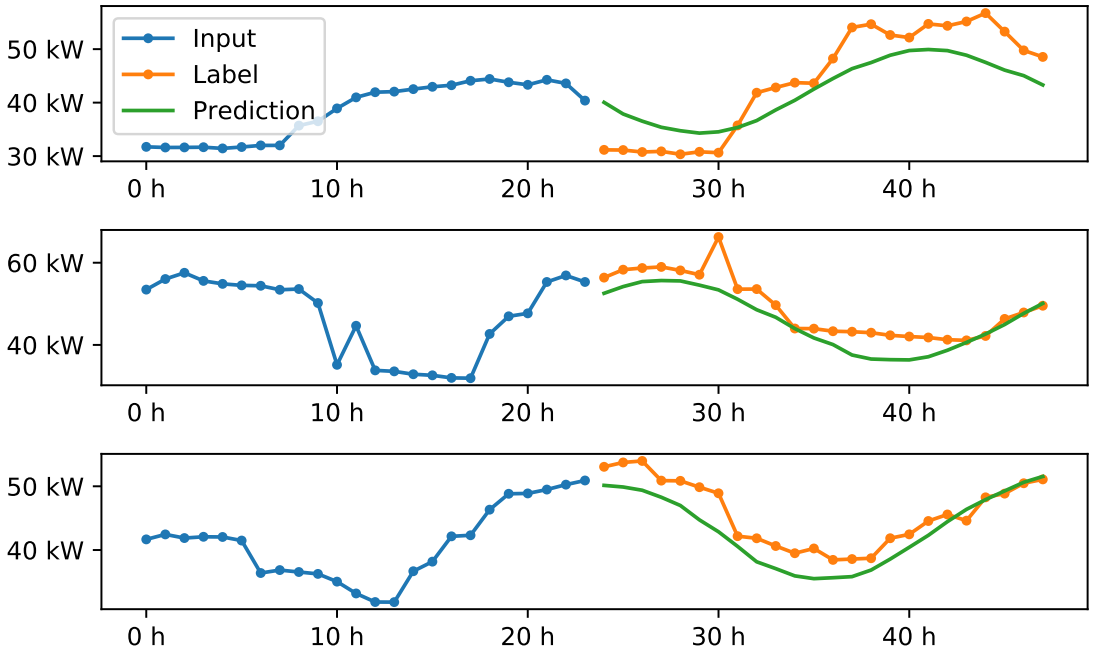

Figure 6.3: Three predictions on example days from the power consumption test dataset with a convolutional NN model with seasonal adjustment and temperature data. This model was trained using a learning rate of 0.0001, a kernel width of 12, 64 kernel filters and no additional dense layers, all of which were determined by the tuner for this model configuration.

agrees with. This means that models are doing more than just remembering training data. The models performed better than the simple baselines in every instance, which is to be expected but it is a good validation nonetheless. The choice of the optimal mode for each configuration made by the validation results is confirmed by the test data.

A summary of the best models for each of the data sources and the results of the simple baselines for comparison can be found in table 6.5 on the next page. The best results using the ANN models are 55%, 54% and 59% better than the repeat baseline for each of the three data sources, respectively. The tuned hyperparameters used to train the models that produced these results can be found in the individual tables of results.

For the electricity price and power consumption data sources the dense model performed the best, while the convolution model was the best performing model for the power generation data source. For the electricity price data however, the AR LSTM models were almost as good as the other two model types.

It is surprising that the AR LSTM models performed worse than the other two simpler model types given that LSTM models are generally thought to be particularly good at this type of problem. One possible explanation for this is that since they are also harder to train, they did not unfold their full potential in these experiments and therefore failed to outperform the other models. Another possibility is that it is very difficult to make

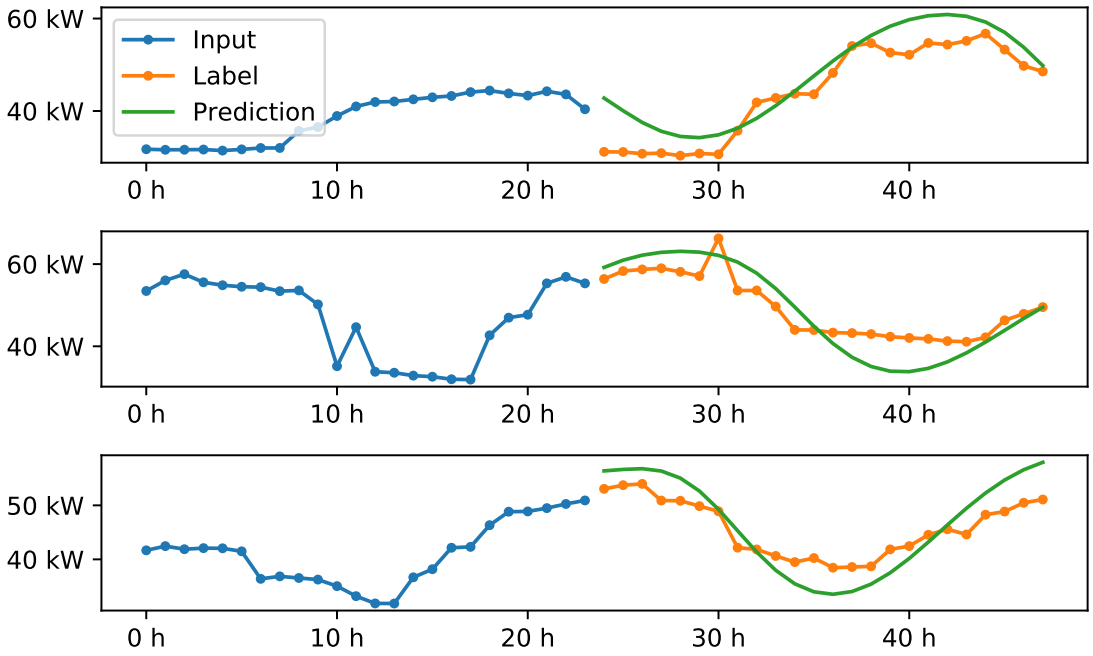

Figure 6.4: Three predictions on example days from the power consumption test dataset with an AR LSTM model with seasonal adjustment and temperature data. This model was trained using a learning rate of 0.00048 and 448 LSTM units, both of which were determined by the tuner for this model configuration.

Data Source	Model Type	Configuration	Mode	Val. MSE	Test MSE
Electricity Price	Baseline $\mathbf{F}_{\text{last}}$	-	-	0.828	0.826
	Baseline $\mathbf{F}_{\text{repeat}}$	-	-	0.744	0.735
	Dense	Seasonal Adjustment	Delta	**0.334**	0.322
	Convolution	Both	Delta	0.390	0.327
	AR LSTM	With Temperature	Delta	0.396	0.330
Power Consumption	Baseline $\mathbf{F}_{\text{last}}$	-	-	1.212	1.676
	Baseline $\mathbf{F}_{\text{repeat}}$	-	-	0.677	0.542
	Dense	Both	Normal	**0.312**	0.506
	Convolution	Both	Normal	0.353	0.411
	AR LSTM	Basic	Normal	0.446	0.493
Power Generation	Baseline $\mathbf{F}_{\text{last}}$	-	-	0.774	1.649
	Baseline $\mathbf{F}_{\text{repeat}}$	-	-	0.930	0.477
	Dense	Both	Delta	0.403	0.269
	Convolution	Both	Normal	**0.383**	0.246
	AR LSTM	With Temperature	Normal	0.433	0.294

Table 6.5: The best configurations for each model type and data source. In each group the best result is highlighted. The results of the two simple baselines are included in this table for comparison.

predictions of this data with the limited additional inputs giving few helpful hints which causes simple models to be more suitable as they are easier to train when there are only simple patterns to predict. Zhang and Kline also observe that when seasonal adjustment is used, simple models perform better than complex ones most of the time [47]. A significant amount of complexity in models is spent on reproducing the seasonalities of data that was not seasonally adjusted.

6.5 Method Evaluation

In order to evaluate the effectiveness of the three methods, namely the delta mode, seasonal adjustment and adding temperature data, the average change in accuracy they cause is measured. For a given method, the average MSE of all training series that used the method is compared to that of all those that did not use it. This average is computed across all data sources and model types. For the delta mode this means all results that used the delta mode are compared to those that used the normal mode.

An average reduction in MSE of 9.85% with seasonal adjustment is measured. The delta mode and the addition of temperature data cause an increase of 3.95% and 5.54% in the measured average MSE, respectively. The chosen best models in the results tables also reflect this as many best configurations include seasonal adjustment. While the same is true for the temperature data, it did not improve the accuracy of all models on average. The temperature data can also be ignored by a model if it does not provide an advantage and its use by the best models may therefore also be incidental. The no input mode causes a 39.66% increase in MSE on average and thereby confirms the usefulness of giving the models inputs at all.

Evaluation of Qualitative Prediction

The qualitative prediction is evaluated using power consumption data with seasonal adjustment and temperature data, based on the convolution model's results. A convolution model was tuned and then evaluated by training a series of 20 model instances, as it was done with the previous experiments. The qualitative prediction is set to use a five minute sample interval for the input and the regular hourly sample interval for the output. The model is trained to predict metrics, in this example the minimum and maximum, calculated over the samples that correspond to each output value.

The tuning resulted in a kernel width of 2, 8 kernel filters and no additional dense layers. Table 6.9 on page 39 contains the measured performance for the models trained with these hyperparameters. An example of a prediction can be seen in figure 6.10 on page 39. Due to the differences between this task and that of simply making regular predictions, the error values cannot be directly compared between them. However, the model does exceed the no input baseline and judging from the example plots, does reasonably well in general. The plots also indicate that the model detects and subsequently predicts a varying distance between the minimum and maximum metrics.

Data Source	Configuration	Dense Units	LR	Mode	Val. MSE	Test MSE
Electricity Price	Basic	272/576/16	0.00010	Normal	0.475	0.548
				Delta	0.470	**0.440**
				No Inputs	0.500	0.504
	With Temperature	1024/16/16	0.00010	Normal	0.511	0.575
				Delta	0.519	0.492
				No Inputs	0.584	0.534
	Seasonal Adjustment	176/16/416	0.00010	Normal	**0.461**	0.527
				Delta	**0.334**	**0.322**
				No Inputs	0.492	0.505
	Both	-	0.00010	Normal	0.528	0.579
				Delta	**0.369**	**0.351**
				No Inputs	0.671	0.599
Power Consumption	Basic	1024/784/16	0.00010	Normal	0.347	0.597
				Delta	0.393	0.750
				No Inputs	0.426	0.834
	With Temperature	-	0.00010	Normal	0.329	0.581
				Delta	0.348	0.650
				No Inputs	0.375	0.661
	Seasonal Adjustment	-	0.00010	Normal	0.325	0.505
				Delta	**0.345**	**0.524**
				No Inputs	0.407	**0.690**
	Both	-	0.00010	Normal	0.312	0.506
				Delta	**0.324**	**0.528**
				No Inputs	**0.368**	0.642
Power Generation	Basic	1024/16/16	0.00010	Normal	0.461	0.290
				Delta	0.464	0.288
				No Inputs	1.119	0.665
	With Temperature	1024/16/16	0.00010	Normal	0.455	0.295
				Delta	0.453	0.291
				No Inputs	1.019	0.664
	Seasonal Adjustment*	-	0.00045	Normal	**0.414**	**0.262**
				Delta	**0.414**	**0.265**
				No Inputs	1.101	0.659
	Both	16	0.00139	Normal	**0.410**	**0.269**
				Delta	**0.403**	**0.269**
				No Inputs	1.011	0.663

Table 6.6: The results of tuning and training the dense NN model. For each data source the three or four best results are highlighted. The third column contains the number of units in each of the dense layers of which there can be zero to four.

Data Source	Configuration	K. Width/Filters	LR	Mode	Val. MSE	Test MSE
Electricity Price	Basic	2/72	0.01235	Normal	**0.430**	0.479
				Delta	0.469	0.579
				No Inputs	0.563	0.539
	With Temperature	2/128	0.00227	Normal	0.568	0.458
				Delta	0.499	0.584
				No Inputs	0.849	0.588
	Seasonal Adjustment	10/32	0.00010	Normal	0.504	0.508
				Delta	**0.402**	0.331
				No Inputs	0.567	0.568
	Both	2/64	0.05121	Normal	0.603	0.573
				Delta	**0.390**	0.327
				No Inputs	0.828	0.602
Power Consumption	Basic	2/72	0.00274	Normal	0.383	0.448
				Delta	0.482	0.692
				No Inputs	0.439	0.479
	With Temperature	2/128	0.00021	Normal	**0.375**	0.425
				Delta	0.482	0.691
				No Inputs	0.407	0.438
	Seasonal Adjustment	12/8	0.00048	Normal	**0.380**	0.462
				Delta	0.454	0.479
				No Inputs	0.466	0.527
	Both	12/64	0.04141	Normal	**0.353**	0.411
				Delta	0.418	0.424
				No Inputs	0.404	0.426
Power Generation	Basic	2/128	0.00364	Normal	0.402	0.259
				Delta	0.708	0.405
				No Inputs	1.002	0.499
	With Temperature	12/128	0.00948	Normal	**0.396**	0.249
				Delta	0.476	0.327
				No Inputs	0.856	0.472
	Seasonal Adjustment*	12/32	0.00055	Normal	**0.388**	0.254
				Delta	0.471	0.336
				No Inputs	0.994	0.504
	Both	12/128	0.00722	Normal	**0.383**	0.246
				Delta	0.470	0.325
				No Inputs	0.870	0.473

Table 6.7: The results of tuning and training the convolutional NN model. For each data source the three best results are highlighted. The third column gives the convolution kernel's width and its number of filters. The configuration marked with * was the only one for which the tuning result included additional dense layers, with 128 and 240 units, respectively.

Data Source	Configuration	LSTM Units	LR	Mode	Val. MSE	Test MSE
Electricity Price	Basic	480	0.01235	Normal	0.545	0.552
				Delta	0.559	0.572
				No Inputs	0.642	0.568
	With Temperature	480	0.00227	Normal	0.669	0.524
				Delta	0.806	0.813
				No Inputs	0.789	0.551
	Seasonal Adjustment	384	0.00010	Normal	**0.502**	0.538
				Delta	**0.396**	0.330
				No Inputs	**0.452**	0.519
	Both	352	0.05121	Normal	0.605	0.613
				Delta	0.551	0.438
				No Inputs	0.560	0.573
Power Consumption	Basic	448	0.00274	Normal	**0.446**	0.493
				Delta	0.498	0.664
				No Inputs	0.581	0.837
	With Temperature	480	0.00021	Normal	**0.449**	0.465
				Delta	0.523	0.653
				No Inputs	0.514	0.582
	Seasonal Adjustment	448	0.00048	Normal	**0.480**	0.438
				Delta	0.574	0.669
				No Inputs	0.723	0.771
	Both	288	0.04141	Normal	0.692	0.874
				Delta	0.487	0.519
				No Inputs	0.616	0.782
Power Generation	Basic	448	0.00364	Normal	**0.443**	0.296
				Delta	0.457	0.306
				No Inputs	0.963	0.501
	With Temperature	512	0.00948	Normal	**0.433**	0.294
				Delta	0.546	0.338
				No Inputs	0.898	0.589
	Seasonal Adjustment	384	0.00055	Normal	**0.448**	0.278
				Delta	0.482	0.295
				No Inputs	0.913	0.496
	Both	480	0.00722	Normal	**0.448**	0.331
				Delta	0.516	0.344
				No Inputs	0.892	0.525

Table 6.8: The results of tuning and training the AR LSTM model. For each data source the three best performing results are highlighted. Two results in the power generation section have the same error value and therefore both are highlighted.

Mode	Validation MSE	Test MSE
Normal	0.482	0.537
No Inputs	0.552	0.581

Table 6.9: The results of training 20 instances of the convolution model to do qualitative prediction.

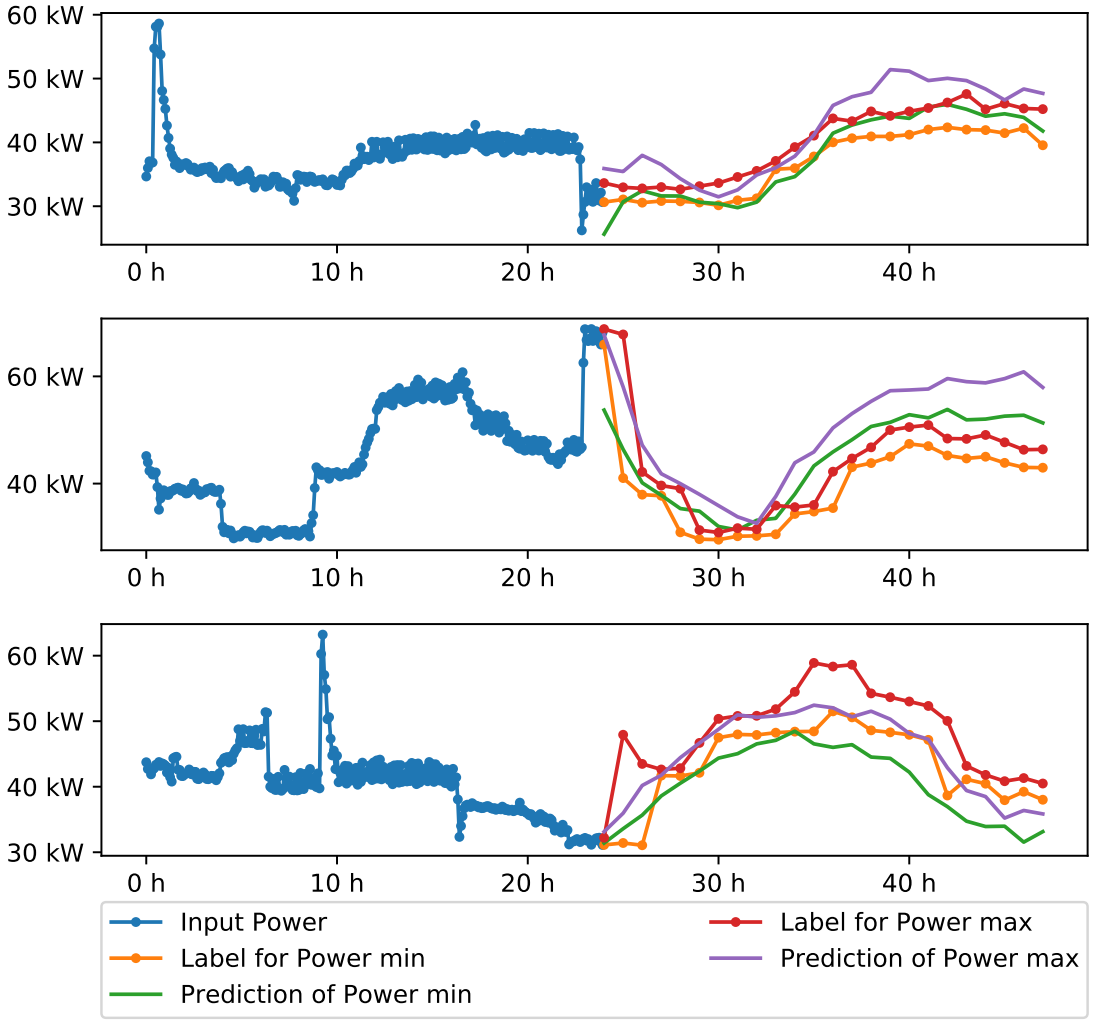

Figure 6.10: Three qualitative predictions on example days from the power consumption test dataset.

7

Conclusion and Outlook

7.1 Conclusion

In this work the application of machine learning models to energy time series data was explored in the context of the ReNuBiL project. First, a formal basis was established encompassing time series, their prediction and the evaluation of such predictions using error metrics. Then, tools for time series prediction such as models using artificial neural networks and statistical performance baselines were introduced. Three different models were used, one with a basic dense neural network, one with a convolutional neural network and, finally, an autoregressive LSTM model.

In order to better schedule electric vehicle bookings, which can be used for VxG while they are connected to a charging station, the project requires predictions of electricity price, power consumption and renewable power generation data. These data sources were then preprocessed by downsampling and normalization. After splitting the data into training, validation and testing sections, a dataset of sequences for use with the models was generated for each of them. A number of methods for improving prediction accuracy were introduced, which were later evaluated in their effectiveness. They are seasonal adjustment, the addition of air temperature data and the delta mode.

The implementation in form of a Python script relied on TensorFlow, the Keras API and Pandas. The University of Lübeck's AI Lab provided access to GPUs including a NVIDIA DGX2 machine that allowed for fast tuning and training of the models. Each model has hyperparameters that heavily influence its performance potential. They are optimized with hyperparameter tuning and the many model instances are trained to determine how well each configuration performs. The configurations are all feasible combinations of the aforementioned methods for improving prediction accuracy.

The many experiments were conducted in an automated fashion and then evaluated as shown in the previous chapter. Using the best configuration and tuned hyperparameters, all models were found to be effective at predicting each data source one day into the future. For the electricity price and power consumption data the basic dense model performed the best with an improvement of at least 54% compared to the repeat baseline, while the convolution model yielded the best results for the power generation data. The evaluation of the three optional methods yielded that seasonal adjustment produced an

average performance improvement, while the other two did not. They were, however, found to be frequently part of the best configurations for each data source. In summary, the application of machine learning methods such as artificial neural networks to energy data yielded results that were better than statistical baselines and the no input baselines for each model. The following section sheds light on the next steps for this work as part of the ReNuBiL project and opportunities for a further improvement of the models.

7.2 Outlook

This work analyzed the application of machine learning to predict future values of three energy data sources. The next step as part of ReNuBiL is the implementation of a service that makes predictions on request using the developed machine learning models and uses them as a basis to control the actions of the scheduler. Such a service would load one or more trained models and then make the requested predictions using the data supplied in each request. While TensorFlow is available for a variety of programming languages and platforms, creating a server for making predictions using Python would be the most convenient. An interesting metric worth evaluating is how much the predictions help the scheduler reach the project's goals compared to if it was using worse predictions or no predictions at all. For an effective comparison to be possible, either a simulation or an accurate measurement of the scheduling behavior without energy data predictions needs to be available. The usefulness of qualitative predictions can be further evaluated when a concrete implementation of the application context, particularly that of the scheduler, is complete.

The models could gradually be improved by re-training or even re-tuning them on a larger and more diverse dataset as time goes on. Training the models with the goal of providing energy data forecasts for real-world use will be different from how they are trained for evaluation of the method itself. More of each dataset can be used, as the three-way split is not required if the best model is already known. Additionally, due to the high variability in prediction accuracy of some models, training multiple instances of the same model and looking at all their predictions together should be considered. A measure of the models' confidence could be calculated from the variation of their predictions.

The power consumption data has a very high resolution coming from the measurement equipment. A more complex but still computationally efficient model could be used to possibly improve prediction accuracy by allowing analyzing high-frequency patterns, which are lost in the downsampled version of the data. A similar improvement could be possible with other sources of additional data, such as separate power consumption measurements for each of the electrical phases, more accurate and diverse meteorological measurements, weather forecasts and possibly even thermodynamic simulations of buildings. Predictions of the energy consumption and demand are made by the grid operator and published on the ENTSO-E Transparency Platform. These predictions, which are likely quite good given the grid operator's knowledge of their own system, could be directly accessed through the platform's API. A possible correlation between the renewable power generation and the electricity prices could also be investigated.

The field of ANN-based machine learning for time series prediction has developed

many methods of improving models. They could also be used to improve the models that are presented in this work. One aspect is regularization, which aims to reduce overfitting, while also accelerating model convergence. Early stopping, which ends training early when a model's performance stops improving for a number of epochs, is already utilized in this work. There are also other ways of applying regularization to a model, such as penalization, which reduces large weights in a network, and dropout [40], which randomly drops units from layers during training. Other methods for enhancing time series forecasting include wavelet and FFT preprocessing, which decompose a time series into its frequency components.

Another option is to use a controlled learning rate schedule for avoiding or even working with local optimal encountered along the way during training [18]. More convolution, pooling and LSTM layers could also be added to the models for further improvements. In particular, the LSTM model, which did not perform better than the simpler models despite its higher complexity and required training effort, could benefit from techniques that improve performance and reduce fixation on local optimal in the search space. The constantly evolving field of machine learning, and artificial intelligence using ANNs in particular, regularly produces new techniques that are applicable to a time series forecasting problem like this one.

Bibliography

[1] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al. TensorFlow: A system for large-scale machine learning. In: *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. Software available from tensorflow.org. 2016, pp. 265–283. URL: https://www.usenix.org/system/files/conference/osdi16/osdi16-abadi.pdf.

[2] Aggarwal, S. and Saini, L. Solar energy prediction using linear and non-linear regularization models: A study on AMS (American Meteorological Society) 2013–14 Solar Energy Prediction Contest. In: *Energy* 78:247–256, 2014. ISSN: 0360-5442. DOI: https://doi.org/10.1016/j.energy.2014.10.012. URL: https://www.sciencedirect.com/science/article/pii/S036054421401161X.

[3] *AI Lab Lübeck*. URL: https://ai-lab.digital-hub-luebeck.de/.

[4] Amasyali, K. and El-Gohary, N. M. A review of data-driven building energy consumption prediction studies. In: *Renewable and Sustainable Energy Reviews* 81:1192–1205, 2018. ISSN: 1364-0321. DOI: https://doi.org/10.1016/j.rser.2017.04.095. URL: https://www.sciencedirect.com/science/article/pii/S1364032117306093.

[5] Bouzidi, L. Wind power variability: Deterministic and probabilistic forecast of wind power production. In: *2017 Saudi Arabia Smart Grid (SASG)*. 2017, pp. 1–7. DOI: https://doi.org/10.1109/SASG.2017.8356514.

[6] Busby, J. W., Baker, K., Bazilian, M. D., Gilbert, A. Q., Grubert, E., Rai, V., Rhodes, J. D., Shidore, S., Smith, C. A., and Webber, M. E. Cascading risks: Understanding the 2021 winter blackout in Texas. In: *Energy Research & Social Science* 77:102106, 2021. ISSN: 2214-6296. DOI: https://doi.org/10.1016/j.erss.2021.102106. URL: https://www.sciencedirect.com/science/article/pii/S2214629621001997.

[7] Cao, J. and Cao, S. Study of forecasting solar irradiance using neural networks with preprocessing sample data by wavelet analysis. In: *Energy* 31(15):3435–3445, 2006. ECOS 2004 - 17th International Conference on Efficiency, Costs, Optimization, Simulation, and Environmental Impact of Energy on Process Systems. ISSN: 0360-5442. DOI: https://doi.org/10.1016/j.energy.2006.04.001. URL: https://www.sciencedirect.com/science/article/pii/S0360544206001009.

[8] Chang, Z., Zhang, Y., and Chen, W. Electricity price prediction based on hybrid model of adam optimized LSTM neural network and wavelet transform. In: *Energy* 187:115804, 2019. ISSN: 0360-5442. DOI: https://doi.org/10.1016/j.energy.2019.07.134. URL: https://www.sciencedirect.com/science/article/pii/S0360544219314768.

[9] Chetlur, S., Woolley, C., Vandermersch, P., Cohen, J., Tran, J., Catanzaro, B., and Shelhamer, E. *cuDNN: Efficient Primitives for Deep Learning*. 2014. arXiv: 1410.0759 [cs.NE].

[10] Chollet, F. et al. *Keras*. https://keras.io. 2015.

[11] *Deutscher Wetterdienst*. URL: https://www.dwd.de.

[12] Dorffner, G. Neural Networks for Time Series Processing. In: *Neural Network World* 6:447–468, 1996.

[13] *European network of transmission system operators for electricity*. URL: https://transparency.entsoe.eu/.

[14] Gundu, V. and Simon, S. P. PSO–LSTM for short term forecast of heterogeneous time series electricity price signals. In: *Journal of Ambient Intelligence and Humanized Computing* 12(2):2375–2385, Feb. 2021. ISSN: 1868-5145. DOI: 10.1007/s12652-020-02353-9. URL: https://doi.org/10.1007/s12652-020-02353-9.

[15] Haque, A. U., Nehrir, M. H., and Mandal, P. A Hybrid Intelligent Model for Deterministic and Quantile Regression Approach for Probabilistic Wind Power Forecasting. In: *IEEE Transactions on Power Systems* 29(4):1663–1672, 2014. DOI: https://doi.org/10.1109/TPWRS.2014.2299801.

[16] HECHT-NIELSEN, R. III.3 - Theory of the Backpropagation Neural Network**Based on "nonindent" by Robert Hecht-Nielsen, which appeared in Proceedings of the International Joint Conference on Neural Networks 1, 593–611, June 1989. © 1989 IEEE. In: *Neural Networks for Perception*. Ed. by H. Wechsler. Academic Press, 1992, pp. 65–93. ISBN: 978-0-12-741252-8. DOI: https://doi.org/10.1016/B978-0-12-741252-8.50010-8. URL: https://www.sciencedirect.com/science/article/pii/B9780127412528500108.

[17] Hochreiter, S. and Schmidhuber, J. Long Short-Term Memory. In: *Neural Computation* 9(8):1735–1780, 1997. DOI: https://doi.org/10.1162/neco.1997.9.8.1735.

[18] Huang, G., Li, Y., Pleiss, G., Liu, Z., Hopcroft, J. E., and Weinberger, K. Q. Snapshot Ensembles: Train 1, get M for free. In: *CoRR* abs/1704.00109, 2017. arXiv: 1704.00109. URL: http://arxiv.org/abs/1704.00109.

[19] Kim, D. Normalization methods for input and output vectors in backpropagation neural networks. In: *Int. J. Comput. Math.* 71(2):161–171, 1999. DOI: https://doi.org/10.1080/00207169908804800.

[20] Kim, T.-Y. and Cho, S.-B. Predicting residential energy consumption using CNN-LSTM neural networks. In: *Energy* 182:72–81, 2019. ISSN: 0360-5442. DOI: https://doi.org/10.1016/j.energy.2019.05.230. URL: https://www.sciencedirect.com/science/article/pii/S0360544219311223.

[21] Ku, Y.-J., Sapra, S., Baidya, S., and Dey, S. State of Energy Prediction in Renewable Energy-driven Mobile Edge Computing using CNN-LSTM Networks. In: *2020 IEEE Green Energy and Smart Systems Conference (IGESSC)*. 2020, pp. 1–7. DOI: https://doi.org/10.1109/IGESSC50231.2020.9285102.

[22] Kumar, D., Mathur, H. D., Bhanot, S., and Bansal, R. C. Forecasting of solar and wind power using LSTM RNN for load frequency control in isolated microgrid. In: *International Journal of Modelling and Simulation* 41(4):311–323, 2021. DOI: https://doi.org/10.1080/02286203.2020.1767840.

[23] Le, T., Vo, M. T., Vo, B., Hwang, E., Rho, S., and Baik, S. W. Improving Electric Energy Consumption Prediction Using CNN and Bi-LSTM. In: *Applied Sciences* 9(20), 2019. ISSN: 2076-3417. DOI: 10.3390/app9204237. URL: https://www.mdpi.com/2076-3417/9/20/4237.

[24] Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., and Talwalkar, A. *Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization*. 2018. arXiv: 1603.06560 [cs.LG].

[25] Liu, Y., Guan, L., Hou, C., Han, H., Liu, Z., Sun, Y., and Zheng, M. Wind Power Short-Term Prediction Based on LSTM and Discrete Wavelet Transform. In: *Applied Sciences* 9(6), 2019. ISSN: 2076-3417. DOI: https://doi.org/10.3390/app9061108. URL: https://www.mdpi.com/2076-3417/9/6/1108.

[26] Lorenz, E., Hurka, J., Heinemann, D., and Beyer, H. G. Irradiance Forecasting for the Power Prediction of Grid-Connected Photovoltaic Systems. In: *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* 2(1):2–10, 2009. DOI: https://doi.org/10.1109/JSTARS.2009.2020300.

[27] McKinney, W. Data Structures for Statistical Computing in Python. In: *Proceedings of the 9th Python in Science Conference*. Ed. by S. van der Walt and J. Millman. 2010, pp. 56 –61. DOI: 10.25080/Majora-92bf1922-00a.

[28] Memarzadeh, G. and Keynia, F. Short-term electricity load and price forecasting by a new optimal LSTM-NN based prediction algorithm. In: *Electric Power Systems Research* 192:106995, 2021. ISSN: 0378-7796. DOI: https://doi.org/10.1016/j.epsr.2020.106995. URL: https://www.sciencedirect.com/science/article/pii/S0378779620307938.

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

[30] O'Shea, K. and Nash, R. An Introduction to Convolutional Neural Networks. In: *CoRR* abs/1511.08458, 2015. arXiv: 1511.08458. URL: http://arxiv.org/abs/1511.08458.

[31] Parkinson, G. Tesla big battery outsmarts lumbering coal units after Loy Yang trips. In: *RenewEconomy*, Dec. 2017. URL: http://reneweconomy.com.au/tesla-big-battery-outsmarts-lumbering-coal-units-after-loy-yang-trips-70003/.

[32] Peng, L., Liu, S., Liu, R., and Wang, L. Effective long short-term memory with differential evolution algorithm for electricity price prediction. In: *Energy* 162:1301–1314, 2018. ISSN: 0360-5442. DOI: https://doi.org/10.1016/j.energy.2018.05.052. URL: https://www.sciencedirect.com/science/article/pii/S0360544218308727.

[33] Pramono, S. H., Rohmatillah, M., Maulana, E., Hasanah, R. N., and Hario, F. Deep Learning-Based Short-Term Load Forecasting for Supporting Demand Response Program in Hybrid Energy System. In: *Energies* 12(17), 2019. ISSN: 1996-1073. DOI: https://doi.org/10.3390/en12173359. URL: https://www.mdpi.com/1996-1073/12/17/3359.

[34] Qing, X. and Niu, Y. Hourly day-ahead solar irradiance prediction using weather forecasts by LSTM. In: *Energy* 148:461–468, 2018. ISSN: 0360-5442. DOI: https://doi.org/10.1016/j.energy.2018.01.177. URL: https://www.sciencedirect.com/science/article/pii/S0360544218302056.

[35] *Reallabor Nutzerorientiertes Bidirektionales Laden*. URL: https://renubil.de/.

[36] Safari, N., Chen, Y., Khorramdel, B., Mao, L. P., and Chung, C. Y. A spatiotemporal wind power prediction based on wavelet decomposition, feature selection, and localized prediction. In: *2017 IEEE Electrical Power and Energy Conference (EPEC)*. 2017, pp. 1–6. DOI: https://doi.org/10.1109/EPEC.2017.8286163.

[37] Schaeffer, R., Szklo, A. S., Pereira de Lucena, A. F., Moreira Cesar Borba, B. S., Pupo Nogueira, L. P., Fleming, F. P., Troccoli, A., Harrison, M., and Boulahya, M. S. Energy sector vulnerability to climate change: A review. In: *Energy* 38(1):1–12, 2012. ISSN: 0360-5442. DOI: https://doi.org/10.1016/j.energy.2011.11.056. URL: https://www.sciencedirect.com/science/article/pii/S0360544211007870.

[38] Shahid, F., Zameer, A., and Muneeb, M. A novel genetic LSTM model for wind power forecast. In: *Energy* 223:120069, 2021. ISSN: 0360-5442. DOI: https://doi.org/10.1016/j.energy.2021.120069. URL: https://www.sciencedirect.com/science/article/pii/S0360544221003182.

[39] Snoek, J., Larochelle, H., and Adams, R. P. Practical Bayesian Optimization of Machine Learning Algorithms. In: *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 2*. NIPS'12. Red Hook, NY, USA: Curran Associates Inc., 2012, pp. 2951–2959.

[40] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. In: *Journal of Machine Learning Research* 15(56):1929–1958, 2014. URL: http://jmlr.org/papers/v15/srivastava14a.html.

[41] The pandas development team pandas-dev/pandas: Pandas 1.3.0. In: July 2021. DOI: 10.5281/zenodo.5060318.

[42] *Time series forecasting*. URL: https://www.tensorflow.org/tutorials/structured_data/time_series.

[43] Wang, J. Q., Du, Y., and Wang, J. LSTM based long-term energy consumption prediction with periodicity. In: *Energy* 197:117197, 2020. ISSN: 0360-5442. DOI: https://doi.org/10.1016/j.energy.2020.117197. URL: https://www.sciencedirect.com/science/article/pii/S0360544220303042.

[44] Wang, K., Qi, X., and Liu, H. Photovoltaic power forecasting based LSTM-Convolutional Network. In: *Energy* 189:116225, 2019. ISSN: 0360-5442. DOI: https://doi.org/10.1016/j.energy.2019.116225. URL: https://www.sciencedirect.com/science/article/pii/S0360544219319206.

[45] Yu, Y., Cao, J., and Zhu, J. An LSTM Short-Term Solar Irradiance Forecasting Under Complicated Weather Conditions. In: *IEEE Access* 7:145651–145666, 2019. DOI: https://doi.org/10.1109/ACCESS.2019.2946057.

[46] Zhang, D. Wavelet Transform. In: *Fundamentals of Image Data Mining: Analysis, Features, Classification and Retrieval*. Cham: Springer International Publishing, 2019, pp. 35–44. ISBN: 978-3-030-17989-2. DOI: 10.1007/978-3-030-17989-2_3. URL: https://doi.org/10.1007/978-3-030-17989-2_3.

[47] Zhang, G. P. and Kline, D. M. Quarterly Time-Series Forecasting With Neural Networks. In: *IEEE Transactions on Neural Networks* 18(6):1800–1814, 2007. DOI: https://doi.org/10.1109/TNN.2007.896859.

[48] Zhao, H. xiang and Magoulès, F. A review on the prediction of building energy consumption. In: *Renewable and Sustainable Energy Reviews* 16(6):3586–3592, 2012. ISSN: 1364-0321. DOI: https://doi.org/10.1016/j.rser.2012.02.049. URL: https://www.sciencedirect.com/science/article/pii/S1364032112001438.

[49] Zhou, S., Zhou, L., Mao, M., Tai, H.-M., and Wan, Y. An Optimized Heterogeneous Structure LSTM Network for Electricity Price Forecasting. In: *IEEE Access* 7:108161–108173, 2019. DOI: https://doi.org/10.1109/ACCESS.2019.2932999.