Data-driven forecasting of electric vehicle charging for frequency regulation

Fredrik Sidh

Gustaf Sundell

LUND UNIVERSITY

Department of Automatic Control

MSc Thesis
TFRT-6169
ISSN 0280-5316

Department of Automatic Control
Lund University
Box 118
SE-221 00 LUND
Sweden

Printed in Sweden by Tryckeriet i E-huset
Lund 2022

Abstract

Electric vehicle (EV) charging may be used in aggregation as virtual batteries to provide a frequency regulating service to the power grid. The service is sold on the Frequency Containment Reserve (FCR) markets, and is traded one and two days ahead. Forecasts of charging patterns are essential to reliably provide this ancillary service. The thesis aims to build a generalized model for forecasting EV charging behavior of 47 EVs in Sweden. The charging behavior is characterized by the state of charge and whether the EV is plugged in to the home charging station or not. Recurrent Neural Networks (RNNs) and XGBoost are applied to produce forecasts that fit the two FCR market settings. Performance of the models is evaluated and compared to a naive baseline in terms of RMSE, MAE, accuracy and F1-score. The naive baseline assumes the same charging behavior as the previous week. The results show both classes of models to consistently beat the naive baseline on both markets, and XGBoost proved to be the best forecaster.

Acknowledgements

We would like to thank our academic supervisor Richard Pates at the Department of Automatic Control at Lund University for his thoughtful guidance and encouragement, and our industry supervisor Daria Madjidian at Emulate Energy AB for his enthusiasm and helpful insights into the problem. We would also like to thank the Swedish National Institute of Computing (SNIC) for providing computational resources, and especially their helpful and patient support team. Finally, we would like to thank all of the EV users who participated in the pilot study and made the project possible.

Contents

Glossary

BCE Binary Cross-Entropy.

CART Classification And Regression Tree.

EV Electric Vehicle.

FCR Frequency Containment Reserve.

FCR-D Frequency Containment Reserve Disturbance.

FCR-N Frequency Containment Reserve Normal.

FFNN Feedforward Neural Network.

GBDT Gradient Boosting Decision Tree.

LSTM Long Short-Term Memory.

MAE Mean Absolute Error.

ML Machine Learning.

MSE Mean Squared Error.

NN Neural Network.

ReLU Rectified Linear Unit.

RMSE Root Mean Squared Error.

RNN Recurrent Neural Network.

SOC State of Charge.

SVK Svenska Kraftnät.

TCL Thermostatically Controlled Load.

1

Introduction

1.1 Aim and Scope

The goal of the thesis is to forecast the charging patterns of Electric Vehicles (EVs). Collections of EV charging may be used in aggregate as virtual batteries to provide frequency regulation to the power grid [Madjidian et al., 2018]. In this setting, the charging patterns are characterized by the two time series of an EV's State of Charge (SOC) and whether it is plugged into the charging station or not.

Individual EV charging patters of the users are modeled through offline training and validation. The forecasts should be able to give reliable predictions per hour of how much frequency regulation can be provided in aggregate, and be generated in time to be traded on the frequency regulating markets. A generalized model of EV charging behavior is built from a set of 47 EVs in Sweden and used to forecast the charging patterns one or two days ahead.

Two classes of Machine Learning (ML) methods are applied: Recurrent Neural Networks (RNNs) and XGBoost; both of which have proven to be promising time series forecasters in literature. The ML methods are deemed to be a suitable approach for several reasons. There is a large number of individual users, where the corresponding time series data is heterogenous in length and contain long periods of missing data. In these settings, conventional statistical methods such as the ARIMA framework become impractical to work with. Leveraging ML methods also allows for learning a generalized model across all multivariate time series of the EVs.

1.2 Background

As the world transitions to more renewable energy sources in the light of climate change, new demand is placed on the control of the electrical grid. With an increased use of renewable energy in the power mix, the supply of energy is becoming more unpredictable due to the inherent dependence on weather conditions for wind and solar photovoltaic power. This creates new challenges of controlling the balance between supply and demand for energy, and thus keeping the frequency of the power

grid stable[IEA, 2022]. In Sweden, this means a frequency of 50 Hz which is regulated through a handful of ancillary services[SVK, 2022c], of which the FCR is the focus of this report.

Virtual batteries

To combat the new control challenges, batteries can be used to instantly dispatch or absorb electricity in order to regulate the frequency of the grid. Instead of using physical batteries, Thermostatically Controlled Loads (TCLs), e.g. heat pumps and air conditioners, and deferrable loads, e.g. charging of EVs, can be used in aggregation to emulate virtual batteries to regulate the power grid[Madjidian et al., 2018]. The flexibility in the virtual battery can then be traded on the FCR market for frequency regulation.

Emulate Energy AB (the solution provider going forward) is a company with proprietary automatic control software for aggregating TCLs and EV charging stations in households as virtual batteries and thus provide a regulating service to the power grid operators through the FCR markets. The solution provider operates in Swedish households by contracts with power providers and appliance manufacturers, to control TCLs and EV charging stations, and uses their proprietary control algorithms to provide this service. In return for lending their devices to the solution provider's control algorithms, the households get an optimized schedule for their electricity consumption[Emulate Energy, 2022].

Out of the possible components of a virtual battery, the focus of the thesis is the use of EV charging. The forecasting aims to model the charging patterns at all time points of the forecasted period. The full forecasted time series may be translated into a measure of flexibility characterized by the time of plug in and the time of plug out from the charging station along with the SOC at the time of plug in. How the three quantities translate to this measure of flexibility is shown in Figure 1.1, although, the details are not pursued in the thesis.

The FCR 1-D and 2-D markets

The FCR market is divided into FCR-N and FCR-D for operations during normal and disturbance respectively depending on how much frequency regulation is required (50.0 ± 0.1 Hz for FCR-N, else 50.0 ± 0.5 Hz for FCR-D)[SVK, 2022c]. The FCR disturbance market is further segmented into up and down regulation. Since January 2022, the Swedish power grid operator, Svenska Kraftnät (SVK), has introduced trade of the frequency down-regulating service (FCR-D down) along with the previously traded ancillary services for FCR-D up and FCR-N[SVK, 2022d].

The quantities for each of the three services are traded on the one day ahead market (1-D) and the two day ahead market (2-D), where the majority of the flexibility is purchased on the 2-D market for FCR-N[SVK, 2022b]. The bids for the 1-D and 2-D markets close at 18:00 and 15:00 respectively[SVK, 2022a]*.

*The Swedish Energy Market Inspectorate (Ei) have at the time of publication accepted the *"En-*

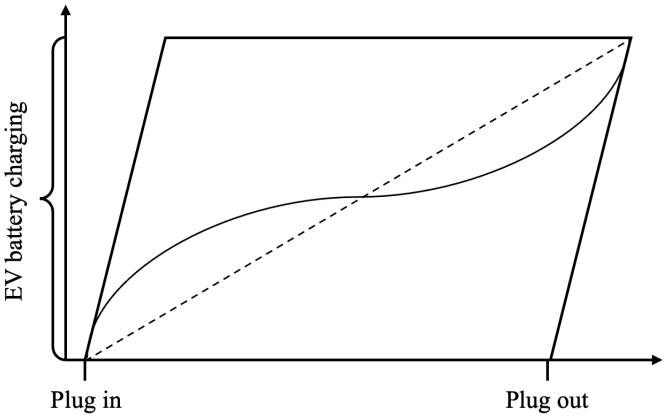


Figure 1.1 Depiction of how time of plug in, time of plug out, and the SOC at time of plug in relate to a flexibility measure. The amount of charging required is shown as the height of the figure. Charging may be schedules earlier or later relative some nominal charging rate. Adapted from [Madjidian et al., 2018].

The bid gate closing time of each of the markets sets the final time point of when a forecast would be needed. For the user of the forecasting model to have time to place a bid on the market, the forecast creation time is set to one hour prior to the closing of bids for each market (i.e. 17:00 for 1-D and 14:00 for 2-D).

Hence, a prediction of the expected aggregate EV charging needs to be produced for each hour of the next day (1-D market) and the day after that (2-D market). Since the charging of an EV can be scheduled earlier or later compared to the forecasted unregulated charging schedule, the charging of an EV can be used for both up and down frequency regulation (as is also shown in Figure 1.1). Hence, the specific FCR market (FCR-N, FCR-D up, FCR-D down) is not differentiated throughout the report.

erginet and Svenska kraftnät proposal for common and harmonised rules and processes for the exchange and procurement of FCR balancing capacity in accordance with Article 33(1) of Commission Regulation (EU) 2017/2195 of 23 November 2017 establishing a guideline on electricity balancing". If further accepted by the EU, the proposal would set the gate closure of bids to 00:30 and 15:00 of one day ahead for what is currently referred to as the 2-D and 1-D FCR markets respectively. The new bid gate closure of the 2-D FCR market would allow for more accurate forecasts, but the main task of forecasting withstands.

1.3 Previous research

EV charging behavior

A large study of 180 000 EVs in Great Britain showed some clear characteristics of the charging behavior of individual users. Charging is typically performed in four different settings: at home, at work, and at slow or fast public charging stations. Naturally, there is also a clear weekly periodicity in the charging behavior where the peak for residential charging is about 19:00-20:00 when people arrive home after work, as well as around 9:00-10:00 when people arrive to work.[Dodson and Slater, 2019]

Forecasting

In the last decade, there has been a massive increase in the application of ML models in countless areas. Various ML methods have been applied to the area of forecasting in relation to EVs. [Hecht et al., 2021] successfully applied Gradient Boosting Decision Trees (GBDTs) and Random Forest Classifiers to forecast EV charging station availability. [Ullah et al., 2021b] applied an ensemble of ML methods (Decision Trees, Random Forests, and K-Nearest Neighbors) to predict the energy consumption efficiency of EVs during use. [Ullah et al., 2021a] also successfully applied GBDTs and Feedforward Neural Networks (FFNNs) to forecast electric consumption of EVs. XGBoost is chosen as one of the applied models due to its successful performance in related fields, even though it does not explicitly encode temporal dependencies.

In contrast, RNNs are capable of explicitly encoding temporal dependencies and are well-suited for a wide variety of time series forecasting problems [Hewamalage et al., 2021]. RNNs have for example been applied to load forecasting of EVs in [Yang et al., 2021].

Out of the wide range of ML models, XGBoost and RNNs represent two methods with fundamentally different assumptions and structures. This thesis aims to evaluate the applicability of these models on forecasting EV charging behavior. To the best of our knowledge, this is the first paper examining the charging behavior of EVs with the goal of frequency regulation through virtual batteries.

1.4 Scientific contributions

This project will help understand how recent developments in ML techniques can be applied to forecasting of EV charging behavior in individual households in Sweden. Specifically, XGBoost and RNN models are used to learn behavior across multiple EVs with the goal of producing reliable forecasts for individual users. Architectures and hyperparameters of the models are optimized for the problem at hand and the results may help researchers and practitioners in related areas discover which models could prove to be important in these problem settings.

1.5 Outline of the Thesis

In Chapter 2 the data is presented in terms of structure, attributes, collection and how it was engineered for the problem. Then in Chapter 3 the forecasting problem is framed mathematically and the theoretical foundations of the models are presented. The approach to applying the statistical methods to the time series problem is then presented in Chapter 4. The results are given in Chapter 5 and then discussed and concluded in the final parts, Chapter 6 and Chapter 7.

1.6 Code

All code relating to the project is publicly available at `https://github.com/sidhson/FCR-forecasting-thesis`. The data used has been removed to respect the privacy of the users.

2

Data

In order to present the data used in the project, we remind ourselves of the goal: to produce reliable forecasts of the deliverable flexibility for the 1-D and 2-D FCR markets. Predicting flexibility on an aggregate level can be derived from the individual flexibility of each EV, and thus the aim of this project is to create forecasts for individual EVs. In order to forecast the flexibility that can be provided by controlling the charging schedule of an EV, knowledge about the charging pattern of the EV is required.

The setting in which the automatic control software can dictate the charge schedule is when the EV is plugged in at the charging station of the owner's home. Thus, the forecast must account for the time of plug in and the time of plug out for the EV, and also differentiate between the home charging station and some other charging station.

In agreement with the service provider, the choice was made to model whether or not the EV is at the home location. Even though there is no strict causal relationship between being at home and plugging in the EV, it is still considered a sufficient design choice. Furthermore, with a reliable forecast of when the EV arrives at the home location, additional modeling could be made of when the EV is likely to be plugged in to the charging station.

Apart from forecasting when the EV is at the home location or not, predicting the EV's SOC at the time of plug in is important in order to know how much the EV needs to be charged and thus the amount of flexibility that can be delivered. An example of the two forecasted time series is shown in Figure 2.1

2.1 Data description

The data gathering process for the project consists in an API provided by the solution provider. The API gathers information on seven attributes from each EV along with time stamps and a minute count at 30 minute intervals*. Tied to each EV is

*This was not always the case. The data had to be resampled to this equidistant sampling frequency to be useful for modeling. The resampling approach is described in detail in Section 2.2

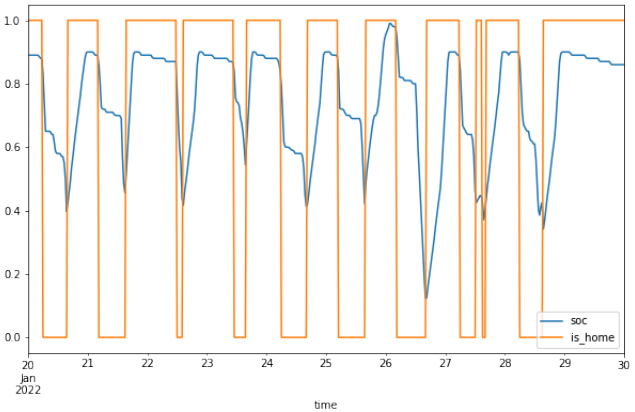


Figure 2.1 An example of the multivariate time series with the two forecasted variables depicted over seven days. A recurring daily pattern of battery depletion and charging occurs when the user leaves and returns home respectively. The depicted time series contains no missing values.

a multivariate time series, which contains the attributes summarized in Table 2.1. Additional attributes were derived from these in order to facilitate modeling; the final attributed used for modeling are summarized in Section 2.3. A more in depth explanation of all the attributes is provided in Appendix A.1. In addition to the time series, each EV also has a unique identification number and some metadata which is further explained in Appendix A.2. Of the metadata available, the home location of the respective EVs was the only variable of interest for modeling purposes.

Attribute	**Description**
`time`	Date-time point in UTC
`period`	Minute tracking
`odometer`	Accumulated driving distance
`latitude`	Position north-south direction
`longitude`	Position east-west direction
`soc`	Battery level of EV
`charge_status`	Charging, not charging or fully charged
`isPluggedIn`	Plugged in to charging station or not
`range`	Possible driving distance with current battery level

Table 2.1 Attributes of the multivariate time series of each EV with a short description. A more detailed description of the data is given in Appendix A.1.

The data collection ran continuously throughout the project, meaning that the available dataset grew as the project progressed. The data was collected starting

2022-01-04, and the last recorded data used for modeling was retrieved at 2022-05-16. As the data collection progressed, more customers joined the pilot study and some also left, resulting in different first and last records of different time series. Dealing with the heterogeneity of the time series is further elaborated on in Section 2.2.

In order to make use of the available data, it needed to be preprocessed, feature engineered and formatted in such a way that the models used could make sense of it. The rest of the chapter explains the preprocessing and feature extraction steps in more detail while the formatting is described in relation to the modeling in Section 4.3.

2.2 Preprocessing of Data

The entire data collection was performed on 53 EVs. Out of these, 5 EVs were not recording their location. The location of the car could potentially be a strong explanatory variable, but more importantly, it is absolutely necessary to determine whether the EV is at home or not. Modeling if the EV is located at home or not is a fundamental goal, and thus these 5 EVs are omitted going forward. In addition, there was one EV which had such few records that it could not be used for modeling, and had to be omitted as well. The data used in the project is thus focused on the time series of the remaining 47 EVs.

The attribute `period` was considered superfluous for the useable data. As the attribute is a minute counter from when the data collection began it does not encode any new information that is not present in the `time` measurement, hence it was deemed unnecessary to store at this point of the preprocessing.

Thus far, the steps up until removing `period` in the preprocessing of the data have been explained. Algorithm 1 outlines all the steps in preprocessing algorithm, the rest of the steps are described in more detail in the remainder of the section on preprocessing.

Interpolation limitations

Missing data is always going to be a factor to face when dealing with real-world data. In the multivariate time series for each EV, an observation must be complete (i.e. each attribute recorded at that time point) for it to be usable. Thus, any observation containing at least one missing value is problematic and needs to be addressed.

When modeling sequences, it is important to have consecutive observations. For small gaps along the time-axis of a univariate time series, some basic interpolation methods can increase the amount of data significantly, without distorting the qualities of the data. As the attributes of the dataset is a mixture of continuous and categorical variables, the appropriate interpolation methods selected were linear interpolation for the continuous attributes, and using the nearest available value for the categorical attributes.

Algorithm 1 High-level algorithm for preprocessing

```
for every EV do
    Retrieve EV data
    if EV is recording location then
        Remove attribute period
        Create piecewise linearly/nearest interpolated series where gaps ≤ 5
        Infer missing categoricals where possible from continuous measurements
        Create new time axis with equidistant 30 min sampling time
        Add distance-column
        Use distance-column to create is_home-column
        Save as processed EV data
    else
        Ignore EV
    end if
end for
```

A gap of five consecutive missing values was considered small enough to bridge using said interpolation methods. The sampling intervals differed somewhat and the maximum interpolated period was a bit over three hours. This maximum period of interpolation was deemed a reasonable trade-off between increasing the amount of usable data and the data still being representative of the generative process. The trade-off for different number of interpolated steps is shown in Figure 2.2.

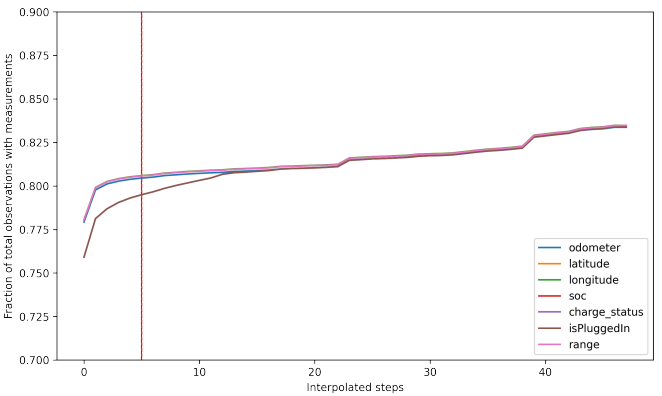


Figure 2.2 Fraction of the total aggregated data which contain measured values (per attribute) with different steps of interpolation. The interpolation steps range from 0 to 49, i.e. up to one full day. The interpolation limit was set to five and is marked in red in the figure.

Interpolation of categorical variables

For the categorical attributes, the same interpolation limit of five consecutive missing values rule was applied and interpolated with the method of nearest available value (selecting the earlier value if two records had equal distance). However, for many EVs, a pattern was observed where the categorical attributes (`isPluggedIn` and `charge_status`) were the only missing measurements for many records. Solely relying on the interpolation approach presented in Section 2.2 means that many rows of data remain without the two categorical variables, which decreases the amount of usable data. However, some causal inference could be made from the continuous variables on the categorical ones. If the EV's SOC had increased between time points, then the EV must have been plugged in at some point between the previous and the next recording. Furthermore, this means that the charge status at that time must have been `CHARGING`. With this approach some extra information could be gained from the original data.

One would think that the vice versa causal inference would be true as well, that if the SOC decreased between two time points, then the EV cannot have been plugged in or charging. Data exploration does however show instances where an EV is plugged in while the SOC decreases (this is also clearly seen in Figure 2.1). This is in line with manufacturers who expect some battery depletion when the car is not in use[†]. As a consequence for the preprocessing, the attribute indicating plug in cannot be derived solely from the decreasing SOC. The charging status could have been computed, but the recording would still be unusable since the plug in status is missing.

As stated earlier, the categorical variables for charging status and whether the EV is plugged in or not are missing more often than the other numerical variables. Since neither of these are target variables, they could have been removed altogether in order to have a larger number of complete records. The difference is however small (as seen in Figure 2.2) and thus it was decided to keep said categorical attributes for their potential explanatory power. Furthermore, more data can also be gathered easily which supports the decision.

Equidistant resampling

In the raw data on the EVs, some variations in sampling intervals were found. This is problematic for models that require equidistant sampling times. The use case of this thesis is to produce forecasts for the charging behavior on an hourly basis, hence 30 minute sampling time was chosen as this gives hour by hour forecasts, but with higher resolution. A new time axis was created with equidistant sampling times of 30 minutes, occurring at whole and half hours (i.e. 12:00, 12:30,...). The data was then projected onto this new time axis following the interpolation scheme outlined

[†]See `https://www.tesla.com/support/range` for more insight into how SOC decreases for an EV not in use.

in Section 2.2. The result is that all data for the usable EVs is at regular sampling intervals of 30 minutes.

Along with these interpolated intervals, some extrapolation was also performed when projecting the original time series onto an equidistant 30 minute sampled time series. If a time point on the new time axis was within 15 minutes from an original record, the extrapolated value was used for that time point.

Feature engineering

At this point the time series have been resampled to equidistant time steps and interpolated in a not too lenient manner. The goal of the forecasting is to predict the EV's SOC along with when it is at home. In order to compute the latter from the data, an attribute called `distance` is added. This attribute is derived from the `longitude` and `latitude` attributes and the metadata containing the home location of the EV. The `distance` attribute is calculated in meters and is necessary as the curvature of the Earth does not allow one to derive the distance to the home solely in the domain of coordinates.

From the new attribute of distance to the home location, the target variable `is_home` can finally be derived. Data exploration examining all time points where an EV is truly plugged in to the charging station at home shows that a distance less than 150 meters to be a good radius.

2.3 Final attributes for modeling

As a concluding remark of the chapter, the time series which have been preprocessed are summarized. Interpolation and resampling of the time series was performed and the attributes `distance` and `is_home` were introduced.

In addition to these new attributes, some final feature extraction was performed. The date-time were encoded as sine and cosine waves with one sine-cosine pair indicating the day of week with a periodicity of a week, and another sine-cosine pair indicating the time of day with a periodicity of a day. The choice of encoding date-time in such a way was to ensure that times close by each other (e.g. before and after midnight) are also close by numerically, as opposed to using a linear scale (e.g. [0,1]). The selected date-time encoding also allowed for far fewer features than if dummy variables would have been used.

Furthermore, the `odometer` measurements were deemed a poor fit for the problem at hand. Firstly, the odometer may differ wildly between individual EVs depending on age of the EV and typical driving distances. Secondly, the `odometer` is strictly non-decreasing over time. As the dataset is split temporally into periods used for learning the model and validation of it, problems arises when a model is fitted to data with a certain distribution which do not generalize to new data. Thirdly, it is hard to motivate why the value of the `odometer` in and of itself could give any information about `soc` and `is_home`. However, the `odometer` could work as a

Attribute	Description
day_sin	Time of day encoding
day_cos	Time of day encoding
week_sin	Day of week encoding
week_cos	Day of week encoding
odometer_diff	Difference of subsequent odometer measurements
latitude	Position north-south direction
longitude	Position east-west direction
soc	Battery level of EV
charging	Binary indicator for charging
not_charging	Binary indicator for not charging
fully_charged	Binary indicator for fully charged
isPluggedIn	Plugged in to charging station or not
range	Possible driving distance with current battery level
distance	The distance in meter to the home location
is_home	Located at the home location or not

Table 2.2 Attributes of the final preprocessed multivariate time series of each EV with a short description. A more detailed description of the data is given in Appendix A.1.

measurement of speed, if the difference is taken between observations, which may say something about the soc. Thus, a new feature odometer_diff was created as the difference between subsequent measurements.

The categorical variable for charging status was also transformed into three dummy variables indicating each of the three states charging, not charging or fully charged. The final attributes used in modeling are summarized in Table 2.2.

3

Theory

This chapter formulates the forecasting problem mathematically and how it fits in to the vast terminology of machine learning. Furthermore, the main classes of models applied in this thesis are introduced, as well as the theory behind them.

3.1 Learning task

Machine learning is used to solve a great variety of problems. However, no solution can be provided without an adequate problem formulation. This section aims to first introduce the forecasting problem in a clear mathematical way. Subsequently, the problem will be tied more closely to practical implementation and to how the learning task of this thesis fits into ML terminology.

Mathematical formulation of the learning task

The learning task is focused on time series forecasting of two measured quantities (target variables going forward, and depicted in Figure 2.1), which may be denoted as $\mathbf{y}^{(k)}(\tau) = (y_1^{(k)}(\tau), y_2^{(k)}(\tau))$, where $y_1^{(k)}(\tau) =$ `soc` of EV k at some time τ and $y_2^{(k)}(\tau) =$ `is_home` of EV k at some time τ. The forecast made by the model is generated at some forecast creation time t and is offset by some integer $D \geq 0$ number of time steps. In the problem setting of this thesis, the forecast should be made in close connection to the bidding time for the FCR market, and apply to the period during which the solution provider will provide frequency regulation, which starts some time after the bid is made, as explained in Section 1.2. A visualization is provided in Figure 4.1. The choice was made that forecasts should be generated one hour prior to the bid gate closure time, meaning that the forecast creation times are set to 17:00 for the 1-D market and 14:00 for the 2-D market. For the forecasted period to begin at midnight, $D = 14$ for the 1-D market and $D = 68$ for the 2-D market, as the time points are evenly spaced by 30 minutes. Furthermore, the forecast consists of predictions at an integer $H > 0$ number of time points. As the forecasted period covers 24 hours for both the 1-D and the 2-D markets, $H = 49$, as

this covers the entire 24 hours as well as the endpoints. Thus, the forecast made at time t will include $2H$ values: $\mathbf{y}^{(k)}(t+D+1), \mathbf{y}^{(k)}(t+D+2), \ldots, \mathbf{y}^{(k)}(t+D+H)$, which is simplified to $\mathbf{y}^{(k)}(\mathcal{H}(t))$ going forward. The forecast has $2H$ values as each $\mathbf{y}^{(k)}$ contains both target variables.

In order to make forecasts, the model will have access to all the information known at the forecast creation time t, which in this setting means all measured values of the EV up to time t. The inputs to a model are commonly denoted as features in ML. With a total of F features, the inputs may be denoted as $\mathbf{x}^{(k)}(\tau) = (x_1^{(k)}(\tau), x_2^{(k)}(\tau), \ldots, x_F^{(k)}(\tau))$, where $x_l^{(k)}(\tau)$ is feature l measured from EV k at time τ. As the target variables are also measured through time, they themselves are included in the input to the forecasting models and two of the features are thus the observed values of the target variables.

The input to the models are, like with the forecasted period, taken from fixed time intervals occurring before the forecast creation time. Which prior time points to include in the input may be chosen with great freedom, and the choices made in this thesis are explained in Chapter 4. For now, the input will be denoted as $\mathbf{x}^{(k)}(\mathcal{T}(t))$.

The models generate forecasts according to their individual algorithms as

$$\mathbf{x}^{(k)}(\mathcal{T}(t)) \quad \xrightarrow{model} \quad \hat{\mathbf{y}}^{(k)}(\mathcal{H}(t))$$

where the distinction is made between values predicted by the model, $\hat{\mathbf{y}}^{(k)}$, and true values, $\mathbf{y}^{(k)}$.

Supervised learning formulation

The forecasting problem may be formulated as a supervised learning task, as a time series can be reformatted into corresponding input and output sequences, from which a model can learn a functional relationship. Given some previous input sequence, the model should make some forecast $\hat{\mathbf{y}}^{(k)}(\mathcal{H}(t))$ the functional relationship may then be learned by comparing against the true values $\mathbf{y}^{(k)}(\mathcal{H}(t))$.

A common way to represent a supervised learning problem is that it is trained on labeled pairs of observations (X_i, Y_i), where $i \in \{1, \ldots, N\}$ and N is the number of observations that the model is trained on. The natural notation then becomes

$$X_i \leftarrow \mathbf{x}^{(k)}(\mathcal{T}(t))$$
$$Y_i \leftarrow \mathbf{y}^{(k)}(\mathcal{H}(t))$$

Since a generalized model is constructed, the formatted data will not carry information on which EV k and which time point t it was generated from as this is unnecessary. The new unique identifier thus becomes i.

Training ML models in a time series setting is different from many other tasks. There are ML models which are capable of explicitly modeling the temporal structure of the data, such as RNNs. Data which explicitly encodes the temporal structure is referred to as having a sequential structure as opposed to a so called cross-sectional structure. A cross-sectional structure assumes all (X_i, Y_i) pairs are independent from (X_j, Y_j) if $i \neq j$. Naturally, this assumption cannot be made for time series, where temporal dependencies make this assumption void. However, some models, such as XGBoost, cannot recognize this temporal dependence and require a cross-sectional encoding of the input and output sequences to (X_i, Y_i) pairs. Still these models have been shown to perform well in time series forecasting. Exactly how the time series are encoded to data structures which are interpretable to the ML models is described in detail in Section 4.3.

There are two target variables to forecast in the supervised learning task. The `soc` is a continuous real value that exists on the interval $[0, 1]$. The `is_home` variable is a binary categorical variable and takes values in $\{True, False\}$. Thus, the learning task is a hybrid of regression and classification.

To encode the categorical values, it is noted that the classification is binary (as opposed to multi-class classification). It is natural to encode the states as $True \rightarrow 1, False \rightarrow 0$. This binary variable may be interpreted as success (the EV is at home) or failure (the EV is not at home).

The rest of the chapter outlines the specific ML models used to solve the time series forecasting problem.

3.2 Neural Networks

A neural network is a learning algorithm which takes some input vector X and builds a nonlinear function $f(X)$ to predict the response Y [James et al., 2021].

Feedforward neural networks

A Neural Network (NN) in it simplest form is a FFNN and consists on an input layer, a hidden layer and an output layer as shown in Figure 3.1. The input layer simply maps each input X_i of the input vector $X = (X_1, ..., X_i, ..., X_p)$ to the next layer. The hidden layer is made up of a set of nodes (the width of the layer) and for each node, it takes some weighted linear combination of the output from the previous layer along with a bias term. The sum

$$h(X) = \beta_0 + \sum_{i}^{p} \beta_i X_i$$

is then run through a nonlinear activation function

$$A = g(h(X)) = g(\beta_0 + \sum_{i}^{p} \beta_i X_i)$$

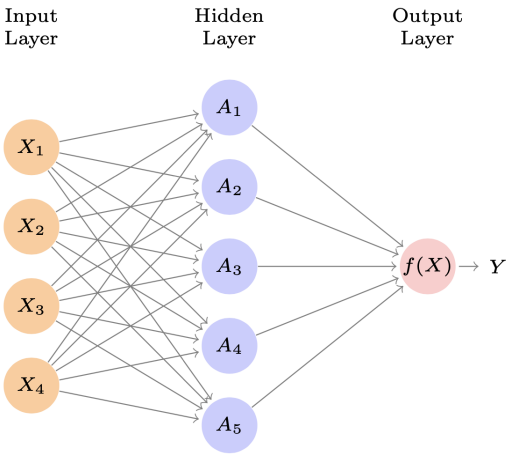


Figure 3.1 Simple Feedforward Neural Network (FFNN) with one hidden layer. [James et al., 2021].

where the activation function can be e.g. a sigmoid function, hyperbolic tangent function (tanh) or Rectified Linear Unit (ReLU). The sigmoid is defined as $\sigma(x) = \frac{1}{1+e^{-x}}$ and takes values in the interval $(0,1)$ and the hyperbolic tangent as $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ and takes values in $(-1,1)$. The ReLU is defined as $\text{ReLU}(x) = \max(0,x)$ and is the most widely applied activation function for hidden layers [Sharma et al., 2020]. The output layer takes the output from the previous layer and potentially applies some function before passing the output of the neural network. The function chosen in the output layer depends on the application and the domain of the response variable, e.g. for binary responses the output from the previous layer can be passed through a sigmoid to map it to $(0,1)$ as it then may be interpreted as an estimated probability.

A FFNN may be extended to consist of more than one hidden layer. Each of the hidden layers then take the output of the previous one and passes the output to the next one. For each node in hidden layer k the output is

$$A_k\left(h_{k,i}(X)\right) = A_k\left(\beta_{k,0} + \sum_i^p \beta_{k,i} h_{k-1,i}\right)$$

In the case of many hidden layers, the NN is said to be *deep* and many nonlinear mappings between the layers means that the learning algorithm can be trained to capture intricate relationships.

Selecting the weights of a NN is a high-dimensional non-convex optimization problem. Training is performed using backpropagation where at each pass over the training data, a loss is calculated according to the current set of weights and then

the weights are adjusted according to the gradient of the loss with respect to each specific weight. The loss function used is dependent on the domain of the response variable, and could be e.g. Mean Squared Error (MSE) for regression and Binary Cross-Entropy (BCE) for binary classification. One way to speed up computations is by calculating the gradient on random batches of observations from the training dataset. The algorithm then adjust the weights by taking a step in the negative direction of the calculated gradient, and then resamples. Once the algorithm has sampled through the entire dataset, it is said to have completed one epoch. The number of samples taken from the training dataset is called the batch size.

In addition to configuring the number of epochs and the batch size, the size of the weight adjustment after calculating each batch gradient, can be set with the learning rate. The rate at which the weights are updated may also be set in an adaptive manner, using momentum or different schedules for updating the learning rate. After training the network, the architecture and the weights constitutes the functional mapping from input vector to the response variable.

Recurrent Neural Networks

RNNs are an extension of FFNNs adapted for sequential data, and particularly exploits the temporal structure of the data. The RNNs consist of an input layer which now takes in the input vector at different time-lagged time points, i.e. $(X_{t-\ell},...,X_{t-1},X_t)$, where ℓ is the number of lags. The next layer is a recurrent layer where each recurrent unit computes and passes on a hidden state which summarizes the previous time-lags. For each recurrent unit the activation is calculated from the input at the current time-lag and the activation at the previous time-lag as

$$A_{\ell,k} = g\left(w_{k,0} + \sum_{j=1}^{p} w_{k,j}X_{\ell,j} + \sum_{s=1}^{K} u_{k,s}A_{\ell-1,s}\right)$$

and the output from the recurrent unit is

$$O_\ell = \beta_0 + \sum_{k=1}^{K} \beta_k A_{\ell k}$$

Note that the weight matrices W, B and U are shared between all time-lags of the recurrent units in the network. The full RNN is depicted in Figure 3.2. In the example, the output is only passed on from the last recurrent output node, but the flexibility of the RNN allows for various architectures, one of which is Sequence-to-Sequence where a full sequence is also the output of the model.

A variation of RNNs is bidirectional RNNs where the hidden states are passed in both directions. As a consequence, information from both future and past time points are available at all time steps, as opposed to the original RNN structure. A as consequence, the number of units and outputs of the bidirectional RNN doubles.[Schuster and Paliwal, 1997]

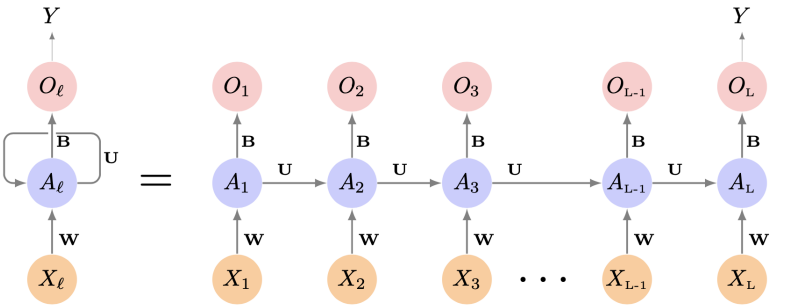


Figure 3.2 Simple Recurrent Neural Network depicted in a compact format on the left and unrolled with the explicit recurrent units (A_l) to the right. A hidden state representation is passed between each recurrent unit in the right depiction. Note that each recurrent unit consists of some width of nodes as in a hidden layer in a simple FFNN. Also note that the weight matrices W, B and U are shared across all recurrent units. [James et al., 2021].

RNNs are trained using Backpropagation Through Time which is similar to backpropagation for FFNNs, but since the weight matrices are shared across time, the loss has to be calculated for each lag and then the accumulated loss is used to update the weights. For long time sequences, many gradients needs to be calculated and hence the training can suffer from vanishing or exploding gradients. To combat this, more robust variations of the RNNs units have been developed.

Long-Short Term Memory Networks

A variation of a simple RNN is the Long Short-Term Memory (LSTM) network introduced in [Hochreiter and Schmidhuber, 1997]. Instead of just passing on a representation of the hidden state, each recurrent unit has a gated architecture consisting of an input gate, output gate and a forget gate. The architecture of the LSTM unit is shown in Figure 3.3 and this updated variation of the RNN is capable of learning long term dependencies much more effectively than the original RNN.

The LSTM keeps track of two hidden states, C_t and h_t, represented by the upper and lower horizontal lines in Figure 3.3 respectively. C_t is only an internal hidden state, while h_t is the hidden state which is also used as output (simliarly to the original RNN). When training the network, the LSTM unit uses the forget gate to learn how much of the previous hidden state (h_{t-1}) to throw away,

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right)$$

the input gate to learn how much of the current input x_t and the previous output h_{t-1} to add to the hidden internal state C_t,

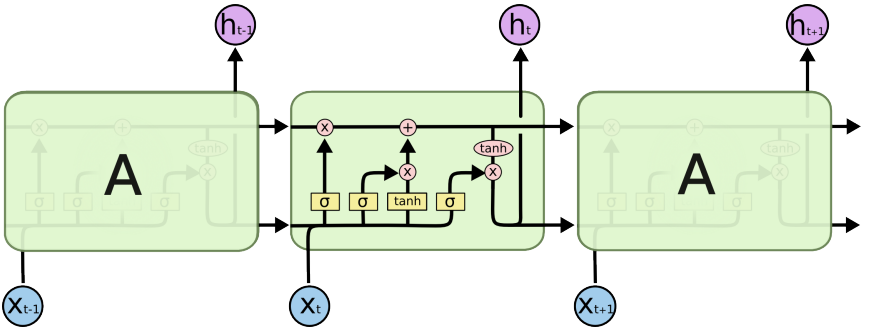


Figure 3.3 Long-Short Term Memory cell. [Olah, 2015].

$$
\begin{aligned}
i_t &= \sigma\left(W_i \cdot [h_{t-1}, x_t] + b_i\right) \\
\tilde{C}_t &= \tanh\left(W_C \cdot [h_{t-1}, x_t] + b_C\right) \\
C_t &= f_t * C_{t-1} + i_t * \tilde{C}_t
\end{aligned}
$$

where $*$ denotes the element-wise product. Finally, the output gate is used to control how much of the hidden states should be returned as output from the LSTM unit as

$$
\begin{aligned}
o_t &= \sigma\left(W_o[h_{t-1}, x_t] + b_o\right) \\
h_t &= o_t * \tanh(C_t)
\end{aligned}
$$

LSTM are chosen as the recurrent unit of the RNN in the thesis due to its comparable performance to other choices of recurrent units.[Bianchi et al., 2017; Jozefowicz et al., 2015]

Regularization and capacity of neural networks

The great flexibility of ML models makes them prone to overfitting and thus requires regularization[Bandara et al., 2020; Lim and Zohren, 2021]. In order to create a reliable NN model, there are many hyperparameters to tweak, which mainly becomes a trade-off between capacity and generalization. A summary of the parameters mentioned here can be found in Table B.1. Increasing capacity means giving the model more freedom to fit the data, and to capture more complex behavior in the learning task. This is accomplished by increasing the number of units in the different layers, as well as the number of layers. The result is more weights and more non-linear functions acting on the input-data, allowing for more closely fitting output.

As the network is trained on a training dataset, increasing the capacity also leads to a higher risk of overfitting. The goal for any ML model is not to perfectly mimic the behavior of the data that it has seen in training, but to do a good job explaining the underlying process that the training data is generated from. Fitting the training dataset too well means that the model might not be able to generalize well on new data from the same data generating process.

To combat this issue, there are different regularization techniques, that aim to make the network generalize better. Kernel regularization means adding a penalty to the loss function proportional to network's weights (e.g. the sum of the L1 or L2 norms), which shrinks the size of the weights. Dropout is another common regularization method, where certain weights of the network are disabled at random during the training process to force other nodes to learn independently. The motivation for these methods are that large weights contribute to overfitting (for kernel regularization) and that by disabling certain weights, other weights need to explain the behavior independently (for dropout).

Furthermore, early stopping may be implemented, where the training process can be terminated if no significant improvement has been made in a certain number of epochs. A good practice is to monitor the loss on the validation dataset. Early stopping can also shorten the training process, which can be practically advantageous when dealing with large networks.

Implementation

Tensorflow 2.5 is a modern library for building NN models[Martín Abadi et al., 2015]. The Keras API provided by Tensorflow is used in this thesis for building the NN models outlined here[Chollet, 2017].

3.3 Gradient Boosting Decision Tree

A GBDT is an ensemble learning method which combines multiple predictors to produce a single prediction. Each of the predictors is a decision tree which splits the prediction space using high-dimensional rectangles as shown in Figure 3.4. The decision tree used is a Classification And Regression Tree (CART) which may be applied to either regression or classification tasks. The prediction of a single CART is then a real valued score, which in the regression setting corresponds to the target variable and in the classification setting may be put through a sigmoid function to produce a class probability. To produce splits between regions, a measure of error improvement from introducing the split is used, e.g. squared error for regression and BCE for binary classification. A split which worsens performance is allowed if later splits in the CART improves the overall result.

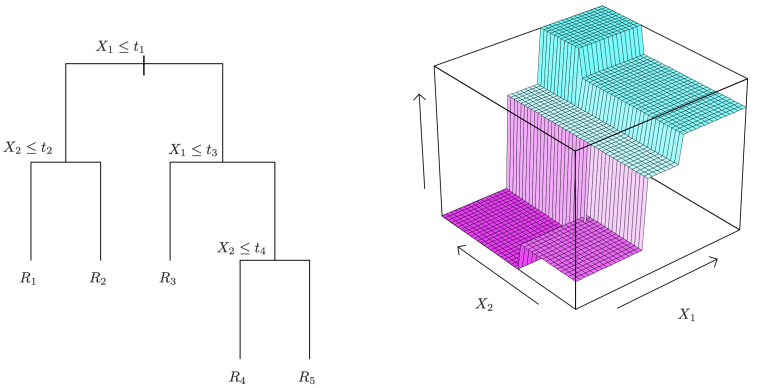


Figure 3.4 A simple decision tree with two features of the prediction space (X_1 and X_2). The high-dimensional rectangles used for predicting the target variable Y (to the right) may be displayed as a decision tree (to the left) [James et al., 2021]

.

The ensemble of CARTs are built by the GBDT algorithm in an additive fashion. The first CART is created in with the approach just described, but subsequent trees are built from the residual of the previous trees. For a GBDT, in addition to the additive construction of the ensemble of CARTs, the prediction of each new tree is optimized using the gradient of the overall loss function (thus, gradient boosting). The prediction formed by the ensemble is then the sum of the predictions formed by all of the model's CARTs

$$\hat{y}_i = \sum_{k=1}^{K} f_k(\mathbf{x}_i)$$

where K trees are used, and each f_k is a CART and $\mathbf{x}_i$ is feature vector. [Friedman, 2001]

DART is an extension of GBDT where dropout is applied to the algorithm in order to combat the issue of overfitting the training dataset. Similarly to how dropout is used in training of deep neural networks, trees of the GBDT may be dropped with some probability. [Rashmi and Gilad-Bachrach, 2015]

XGBoost (eXtreme Gradient Boosting) is a modern framework which implements GBDT and DART. The XGBoost implementation of the GBDT is somewhat different in that each tree is additionally regularized with L2 regularization. The splits and the adding of new trees is done in an efficient manner and trees' importance are weighted by how much they improve the model's performance. [Chen and Guestrin, 2016]

Furthermore, XGBoost has been celebrated as the implementation provides a flexible interface where tuning and training models is simple. XGBoost models are

also quickly trained, thanks in large part to its efficient algorithm for finding splits.

Multi-output multi-horizon adaption of XGBoost

In its initial formulation, XGBoost is a model that only outputs a single value. In the binary classification setting, the model can either output a class prediction (for the class with the highest probability) or the probability itself for the positive class. In the regression setting, the output is just the continuous value predicted. This is still true in the current implementation of XGBoost[XGBoost, 2022a], however efforts have been made to extend the model to multiple outputs[Zhang and Jung, 2019]

The decision was made to use the current implementation with single output. As a consequence, further architecture needed to be built for the XGBoost model to perform the multi-horizon forecasting task. As the forecasting problem requires models to output two separate target variables at several time steps, the solution was to create a collection of XGBoost models for each target variable and time point in the forecasting period. With a fixed length of the forecasting period, H, the result was a collection of $2H$ models, half of which were regressors and half classifiers.

Regularization and capacity of XGBoost

Just as with NNs, XGBoost models can be tuned to increase capacity and avoid overfitting. The number of trees the model builds, as well as to what maximum depth (levels in the trees) are parameters that increase the capacity of the model and thus also increase the risk of overfitting. What underlying booster is used may also be altered, where the variant DART has the added drop-out behavior which works to prevent overfitting.

Several other parameters exist to help regularize the model, all summarized in Table B.2. The default XGBoost model actually already incorporates L2-regularization, analogously to the one described for neural networks. Further regularization is achieved with the gamma-parameter which sets the minimum improvement that a tree must contribute to the loss function to be added to the model. The learning rate (shrinkage factor) can also be used to adjust the speed at which new trees reduce the residual error of the predictions.

The regularizing parameters more unique to XGBoost are perhaps the subsample and column sample ratios. If these ratios are set to less than one, the individual trees are shown a random subset of observations and columns (features) respectively. The aim to make the model generalize better since potentially dominating features or observations are not show to all of the decision trees.

4

Methods

This chapter presents the models applied to the forecasting problem. The formatting of data for the models is described, and then how the models were configured, optimized and trained. Lastly, the performance metrics for evaluation of the final models is presented.

4.1 Forecasting in the FCR market setting

The forecasting task was discussed from a theoretical point of view in Chapter 3. In terms of the application of the model, the forecasting is conducted in two settings: on the 1-D and the 2-D FCR markets respectively. With the different bid gate closing times of the two markets, one model is built for each market. This is in order to use as much of the available data as possible.

It would be possible to model these markets with one larger model which outputs forecasts for both of the days, but this would mean that less information is used in the predictions for the 1-D market since the forecast would have to be generated in time for the closing of the 2-D FCR market bids. Still, the focus of the modeling, in terms of architecture and hyperparameter tuning, is of the 1-D market to ensure that reliable forecasts are possible at all. Then the successful models from the 1-D market are re-trained and applied to forecast the 2-D market as well.

The common idea for the forecasting problem is that the forecasting model may use all information available up to and including the time point t (when the forecast is created), to make a forecast during the coming period of interest. In the use case setting, forecasts need to cover the coming day (or the next), starting and ending at midnight. Thus, the hours between forecast creation time (14:00 or 17:00) and midnight of the forecasted day are not of interest. Given that the sampling frequency is 30 minutes, an interval of 49 time points covers a 24 hour period, and includes both endpoints (midnight to midnight). Figure 4.1 shows an example of a time series with the forecast creation time and forecasted period highlighted, as well as the input sequences, which are explained in detail in Section 4.3.

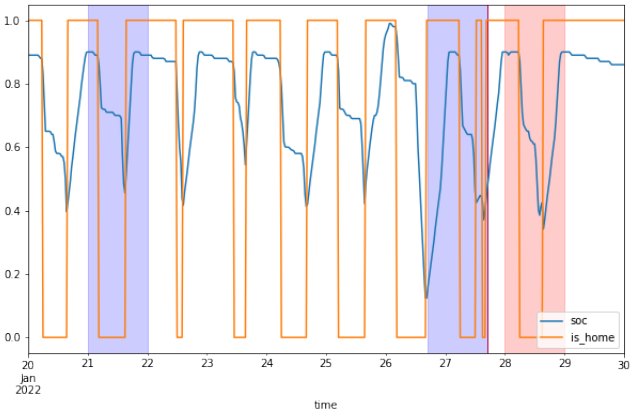


Figure 4.1 An arbitrary time series of the two target variables with the periods of input data (marked in blue) and the mapped output data (marked in red). The input and output time points depict the FCR 1-D market setting where the patch of unseen data stretches from 17.00 until midnight. The forecast creation time is shown by the red vertical line. Note that additional time series are used as input to the model.

4.2 Forecasting Models

The theoretical background of the statistical models applied to the problem were introduced in the previous chapter. The specific models applied are now described, first briefly and then in more detail.

Since the task was to create forecasts on individual EVs, one might imagine that individual models are trained and evaluated for each of the unique EVs. Notably, this would mean fine tuning a great number of models, as well as training each model on a small amount of data. To better suite the scope of this thesis, this approach was ruled out.

Alternatively, one might imagine clustering the EVs into a small number of groups that share important statistical qualities. This would reduce the number of individual models in need of fine tuning, and increase the amount of data each model is trained on. Preliminary data exploration showed that the time series of the target variables did not differ enough to motivate modeling of clusters of EVs with similar charging patterns. The motivation of this choice is however outside of the scope of this report.*

The resulting conclusion and limitation was that the modeling would be conducted on all EVs, aggregating their data as a single generalized EV. It should however be noted that at all times the input to the model was always from the same EV as the corresponding output, and that the data was mixed in such a way that the

*For a thorough review on the concept of clustering multivariate time series, see [Shifaz et al., 2021]

original EV identification was retrievable.

The models built use a direct forecasting strategy of outputting the full multi-step forecasting horizon at once, as opposed to a recursive approach of feeding back each single new prediction. The choice was made as the direct approach has shown to be superior[Wen et al., 2017; Taieb et al., 2011].

The models evaluated in this report for the forecasting task are:

- RNN
- XGBoost
- Naive baseline

The models are first briefly described before going more into depth on the RNN and XGBoost models in terms of data formatting, training, optimization and so forth.

RNN

The RNN models applied to solve the forecasting task allow for joint forecasts to be made for the two target variables `soc` and `is_home`. Performing joint forecasting with a single model allows one to reduce the need for optimizing architecture and hyperparameters from two models to one. The same input data was thus used for the forecasts of each of the target variables. Individual modeling of the two target variables was also examined, but this did not generate any improvement and the advantage of a single optimized joint model was deemed more important.

XGBoost

The XGBoost models are inherently single output models, and as described in Section 3.3, a collection of models was necessary, one for each target variable and for each forecasted time step. With this in mind, it did not make sense to reason about joint forecasting as in the case with RNNs.

Naive baseline

The last model is the baseline which all other models are compared against. The naive baseline looks at the forecasted day one week prior and assumes the same behavior for each time step. By shifting the forecast one week, the different behaviors during weekday and weekend are accounted for, but still a very simple baseline is used.

A different reasonable baseline would be to expect the same behavior as the last 24 hours. There is however a fundamental problem with this hypothetical baseline as the forecasting setting requires the forecast to be created a few hours prior to the time of the actual forecast. Thus the last few hours of the day would not be available at the forecast creation time, and the forecast would be impossible to provide. Additionally, a baseline like this would not have accounted for the weekly periodicity of charging behavior.

4.3 Data formatting for learning

In order to train the models for the forecasting task, the data presented in Chapter 2 had to be reformatted into a training dataset with input-output pairs (X_i, Y_i) for the supervised learning setting as described in Section 3.1. After formatting, the full dataset consisting of aggregated (X_i, Y_i) pairs is denoted $(\mathbf{X}, \mathbf{Y})$.

After the preprocessing stage, the time series data now had the same fundamental multivariate tabular format as before the preprocessing. However, each EV may still have gaps in their time series which were deemed too large to bridge during interpolation. Also, the EVs have varying length time series, as recording of data started later and stopped for some vehicles.

The set of (X_i, Y_i) observations is generated by sliding a fixed length window which maps parts of the time series to each pair (X_i, Y_i). The sliding window tries to maximize the number of observations and thus allows for the observations to overlap. It is when this sliding window is applied that data containing missing values is ignored.

The forecasted period in relation to the forecast creation time was detailed in Section 3.1, however the window creating input was not. As mentioned, the choice of how to create this window is very free. The decision was made to create a two-part window for the inputs to the models: the 24 hours leading up to the forecast creation time ("yesterday") and the 24 hours from one week prior to the forecasted period ("last week"). An example of the input sequences and forecasted period is shown in Figure 4.1.

In relation to the forecast creation time t, these intervals can be described analogously to how the forecasted period was described in Section 3.1. With $L = 48$, the yesterday-window includes time points $t-L, t-L+1, \ldots, t$. With $B = 7 \cdot 24 \cdot 2 = 336$ being the number of 30 minute time steps in a week's time, the last week-window includes time points $t+D-B, t+D-B+1, \ldots, t+D-B+H$, where D depends on whether the data is for the 1-D or 2-D market.

The choice of window was based on a consideration of three aspects, in order of relevance: relevant information, data loss and model complexity. It was reasoned that very recent information should be valuable, as it pertains closely to the imminent forecasted period, hence the yesterday-sequence. As for the last week-sequence, the choice was made on the reasonable assumption that EV charging behavior tends to repeat on a weekly basis, as supported by [Dodson and Slater, 2019]. As for the data loss, too large windows means losing more data as a consequence of missing values occurring in the sequence (a full week of input data would have resulted in a 22 percent data loss and was thus ruled out). Furthermore, large input windows increases a model's complexity and make it more difficult to train if no additional measures are applied. The two input sequences were used for all modelling, however variations of only using either last week or yesterday were also tried.

As portions of all of the time series are traversed by the sliding window, the

resulting (X_i, Y_i) pairs can be taken from any EV k and any forecast creation time t and aggregated to the resulting dataset $(\mathbf{X}, \mathbf{Y})$. Another benefit of the approach is the indifference to the heterogeneity of the lengths of the time series for different EVs.

How the dimensions of features and time are formatted to trainable data depends on the model applied and can be categorized in two main ways: sequential structure or a cross-sectional structure. These two data formatting approaches are presented in the two subsequent sections.

Sequential structure

The RNN models use sequentially structured data where the data is formatted as input- and output-sequences with preserved time-dimension. This structure is conveniently represented as matrices:

$$X_i^{(last\ week)} = \begin{bmatrix} x_1^{(k)}(t+D-B) & x_2^{(k)}(t+D-B) & \dots & x_F^{(k)}(t+D-B) \\ x_1^{(k)}(t+D-B+1) & x_2^{(k)}(t+D-B+1) & \dots & x_F^{(k)}(t+D-B+1) \\ \vdots & \vdots & \ddots & \vdots \\ x_1^{(k)}(t+D-B+H) & x_2^{(k)}(t+D-B+H) & \dots & x_F^{(k)}(t+D-B+H) \end{bmatrix}$$

$$X_i^{(yesterday)} = \begin{bmatrix} x_1^{(k)}(t-L) & x_2^{(k)}(t-L) & \cdots & x_F^{(k)}(t-L) \\ x_1^{(k)}(t-L+1) & x_2^{(k)}(t-L+1) & \cdots & x_F^{(k)}(t-L+1) \\ \vdots & \vdots & \ddots & \vdots \\ x_1^{(k)}(t) & x_2^{(k)}(t) & \cdots & x_F^{(k)}(t) \end{bmatrix}$$

In the setting where both input sequences are used, X_i refers to both $X_i^{(last\ week)}$ and $X_i^{(yesterday)}$. The two input sequences are fed separately to the model. The Y_i are constructed in a similar manner:

$$Y_i = \begin{bmatrix} y_1^{(k)}(t+D+1) & y_2^{(k)}(t+D+1) \\ y_1^{(k)}(t+D+2) & y_2^{(k)}(t+D+2) \\ \vdots & \vdots \\ y_1^{(k)}(t+D+H) & y_2^{(k)}(t+D+H) \end{bmatrix}$$

Now, the (X_i, Y_i) pairs have been defined, the complete dataset when formatted with a sequential structure is described by $\mathbf{X}$ and $\mathbf{Y}$. One can think of these entities as matrices stacked in a third dimension which is the observations.

Cross-sectional structure

In contrast to the sequential structure, the cross-sectional structure does not preserve the temporal structure of the data. Instead, XGBoost models which utilize the cross-

sectional structure views each lagged feature as independent. The data is structured as N observations in the same ways as previously, but the dimensions of time and features have been coalesced into a single dimension. This is shown in

$$X_i = \begin{bmatrix} x_1^{(k)}(t+D-B) \\ x_2^{(k)}(t+D-B) \\ \vdots \\ x_F^{(k)}(t+D-B) \\ \vdots \\ x_1^{(k)}(t+D-B+1) \\ \vdots \\ x_F^{(k)}(t+D-B+H) \\ x_1^{(k)}(t-L) \\ x_2^{(k)}(t-L) \\ \vdots \\ x_F^{(k)}(t-L) \\ x_1^{(k)}(t-L+1) \\ x_2^{(k)}(t-L+1) \\ \vdots \\ x_F^{(k)}(t-L+1) \\ \vdots \\ x_1^{(k)}(t) \\ x_2^{(k)}(t) \\ \vdots \\ x_F^{(k)}(t) \end{bmatrix}$$

where each feature is lagged according to the yesterday and last week time points. If one of the two sequences were to be excluded, this range of lags would be exempt from the vector. All time points and features are viewed as independent and placed on the same dimension, which is the important difference from the sequential structure.

Paired with each input X_i is the corresponding target vector,

$$Y_i = \begin{bmatrix} y_1^{(k)}(t+D+1) \\ y_1^{(k)}(t+D+2) \\ \vdots \\ y_1^{(k)}(t+D+H) \\ y_2^{(k)}(t+D+1) \\ y_2^{(k)}(t+D+2) \\ \vdots \\ y_2^{(k)}(t+D+H) \end{bmatrix}$$

with the two target variables shifted forward $D+1$ to $D+H$ steps.

Importantly, for the cross-sectional structured data each input vector is matched with an output vector. This mapping of single dimension input-output pairs allows the whole dataset $(\mathbf{X}, \mathbf{Y})$ to be stored in a single two dimensional tabular format.

Data split for modelling

Throughout the thesis, the data used was split into the three datasets training, validation and testing, with a split of 0.6, 0.2 and 0.2 respectively. The training dataset is used by the model to learn optimal parameter values through the learning algorithm. The model's performance is then evaluated on the validation dataset, yielding a measurement on how well it generalizes, which gives information on potential improvements. The test dataset is left to the final results. The split is made along the time axis, i.e. the test dataset comes from data gathered later than the validation and training datasets. The split was made for each EV, after formatting and before aggregation. The result was that all EVs are represented in all three datasets proportional to the relative amount of data available for each EV and not based on some specific time point. However, in general, most EV are synchronized in time.

With said split of the full dataset, the training set will generally contain data from winter and early spring while the validation and testing sets contain data from later during the spring. The distribution of the aggregated observations is shown in Figure 4.2.

Feature-wise normalization of data

A common practice in ML is to normalize the data at feature-level. ML methods that use gradient descent-based training algorithms can under-perform when being fed data with very large values or with data where the different features generally come in different orders of magnitude. This increases the risk of the learning algorithm to take far bigger leaps than ideal and that larger magnitude features drown out the influence of smaller ones[Chollet, 2017].

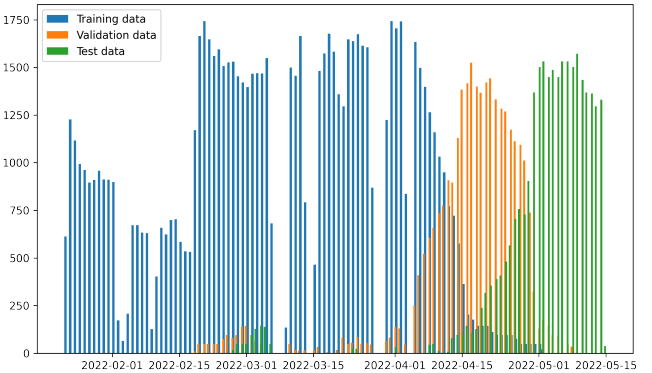


Figure 4.2 Distribution of the observations of each of the datasets for training, validation and testing. The EV time series are heterogeneous both in terms of first and last records and the distribution of missing values throughout the time series. Since each EV must be modeled, the fraction of observations in each time series was set to be the split between different datasets which resulted in the distribution of observations depicted here.

Thus, the goal is for the model to be trained on data where all features take values in approximately the same range, and that range should be quite small, e.g. $[-1,1]$ or $[0,1]$. Because the statistical distribution of the data should be preserved, simple affine transformations are usually applied to the data.

The choice was to use Min-Max-Scaling, where each feature is normalized to the interval $[0,1]$ by

$$X' = \frac{X - X_{min}}{X_{max} - X_{min}}$$

i.e. each feature is normalized by the observed minimum and maximum values. Although tree-based methods are not as sensitive to normalization as Neural Networks, the scaling was done feature-wise for all the models.

In line with the data split described in Section 4.3, the validation and test sets should not be "seen" prior to the models being trained. This applies to normalization as well, thus the minimum and maximum values observed in the training set were used to normalize the validation and test datasets as well. Note that the target variables were never normalized, as `soc` is bounded in $[0,1]$ by definition and `is_home` is a binary variable.

4.4 Model training

The overall goal of modeling the forecasting of a specific model is to find a model which given some training data, should generalize well to unseen data in the form of a validation dataset. Thus, the goal was to train the parameters of a model and then determine how well it generalizes by computing a loss with respect to the validation data. To train each model to solve both the classification and regression tasks, suitable loss functions needed to be selected.

Loss function for RNN models

For the training of a neural network, a loss function is needed to calculate the gradients during backpropagation. Each gradient is calculated as an approximation of the true gradient by looking at a subset of the data, called a batch.

For the regression task of predicting the SOC of the EV, the Mean Squared Error (MSE) was used during training to tune the weights of the model,

$$\mathcal{L}_{MSE} = \frac{1}{N_{batch}} \sum_{i=1}^{N_{batch}} (\hat{y}_i - y_i)^2$$

where N_{batch} is the total number of observations in the batch, $\hat{y}_i$ is the estimated `soc` and y_i is the true `soc`.

For the binary classification task of predicting whether the EV is located at the home or not, Binary Cross-Entropy (BCE) was used during training,

$$\mathcal{L}_{BCE} = -\frac{1}{N_{batch}} \sum_{i=1}^{N_{batch}} y_i \cdot \log \hat{y}_i + (1 - y_i) \cdot \log (1 - \hat{y}_i)$$

where y_i is the true label and $\hat{y}_i$ is the predicated probability of the positive label.

For a single neural network model forecasting multiple steps of two quantities (i.e. two channels, one for regression and one for binary classification), the loss functions must be summarized in a single loss quantity. For the two different loss functions of each output channel (MSE and BCE), the loss was calculated for each time step of the forecast and the summarized into a scalar value. The loss of each channel was then combined by scaling the MSE loss, as after it was found that the order of the MSE was about a tenth of the order of the BCE. The single measure of the loss was calculated as

$$\mathcal{L}_{total} = 10\mathcal{L}_{MSE} + \mathcal{L}_{BCE}$$

Loss function for XGBoost models

The XGBoost leverages the cross-sectional data structure to predict a single value. During training, trees are added depending on how well they reduce the loss function. The loss functions used to train the XGBoost model was BCE for the classification task and Squared Error (SE) for the regression. The Squared Error is defined

in the same way as the MSE, with the simple distinction that the sum is not divided by the number of observations. In an optimization setting, minimizing SE is equivalent to minimizing MSE.

4.5 Hyperparameter tuning

The ML models explored in the thesis each require some tuning of hyperparameters and architectures in order to generalize well to new data. The XGBoost models only requires tuning of hyperparameters while the RNN models, in addition to hyperparameters, also requires tuning of the model architecture.[†]

The hyperparameter search used is a random search. For each hyperparameter, a set or distribution of values are given from which a value is sampled at each run of the model. After running a large number of models, a pattern should hopefully start to emerge where some configurations perform better than others. Hence, the search space of the hyperparameters can be reduced and further tuning can be performed [Bergstra and Bengio, 2012].

RNN optimization procedure

Both architecture and hyperparameters were varied to try to find the best performing model. The search procedure of model exploration consisted mainly of the following steps. First, selecting an architecture and a large search space of hyperparameters for the model. Second, training models until an indication of the importance of each hyperparameter with respect to the loss on the validation dataset. Finally, re-iterating the first and second step by decreasing the search space of hyperparameters.

This procedure was repeated for a large number of architectures. The fundamental architectures tried and hyperparameter search space are outlined in the next two sections.

RNN architecture

A great number of architectures were evaluated for the RNN models, all of which used the LSTM unit as recurrent unit.

The simplest RNN architecture consists of a single LSTM layer which inputs some feature vector at each time step and then outputs the predicted target variable directly from each recurrent unit. Extensions of this model can be made by vertically stacking multiple LSTM layers to accomplished higher level representations of the data. The output of each recurrent unit in the LSTM layer is then the input of the next layer. As for the output, it is useful to apply some learned affine function to the output of the recurrent units at each time step. It is also possible to follow

[†]Literature may refer to the RNN model's architecture as hyperparameters, but in this thesis the architecture is separated.

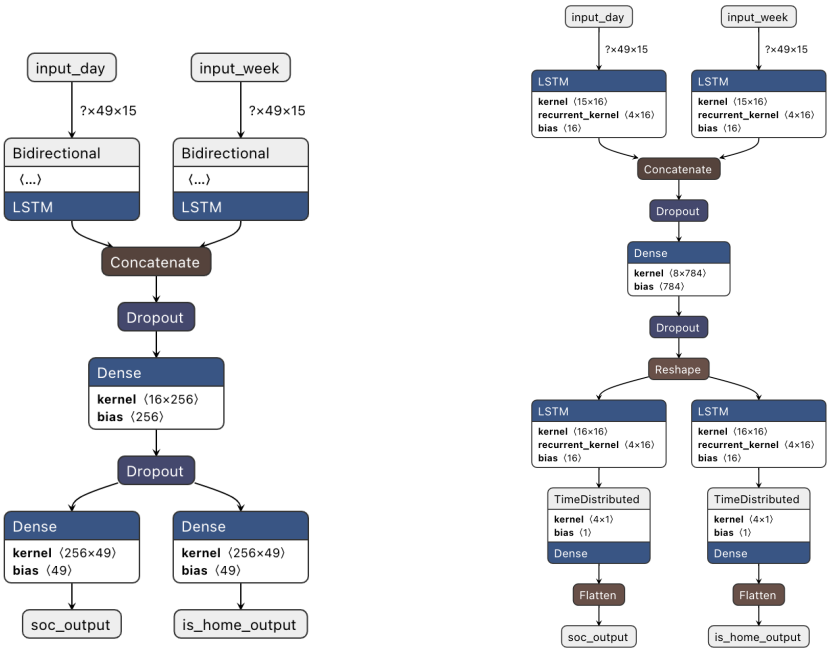
the LSTM layer(s) by some larger fully connected layer which outputs the whole forecast[Hewamalage et al., 2021].

Another class of RNN architectures are encoder-decoder architectures where the first part of the network, the encoder, aims to encode all relevant information of the data into a single vector. The second part of the network, the decoder, then aims to decode said vector to form the forecast. These two parts of the network generally consist of single or stacked LSTM layers[Hewamalage et al., 2021].

Out of all the RNN models, the best performing models in terms of architecture were models leveraging LSTM units to encode the temporal dependence of the input sequences, combined with some fully connected layers with high complexity which could learn from the encoded input time series. Some slight variations of these were the models RNN-oFF and RNN-oREC which used different neural network structures to generate the forecasted sequence from the encoded time series. The architecture of the models used are shown in Figure 4.3. The names of the two models stem from where they differ: `oFF` signifies that the model has a standard Feed Forward layer with the same number of nodes as the number of forecasted time points as output for the target variables separately, whereas `oREC` signifies that the output is generated by an recurrent LSTM layer that outputs sequences, with a final standard activation node for each forecasted time point and target variable. For the RNN-oREC model, the output from the previous layer is reshaped to resemble a sequence, which the final LSTM layer can take as input. Hence the number of units in the final FF layer in the RNN-oREC models needs to be a multiple of 49, the number of time steps in the generated forecast.

RNN hyperparameters

When exploring the performance of varying neural network architectures, the hyperparameters were also varied. Initially, with a larger hyperparameter space, for each run of a specific model, the number of epochs (i.e. the number of times the training dataset was passed over) was set to 5 000. Along with this setting, early stopping was used with a minimum improvement of 0.0001 and a patience of 10 epochs without any improvement of the loss on the validation data. A great number of hyperparameters were evaluated at this point, although, these are not of main interest for the report. The main outcome from the initial search was that the two architectures shown in Figure 4.3 were selected, and that the choice of activation for the `soc` output was set to sigmoid. As the `soc` naturally takes values in $[0, 1]$, the sigmoid was tried in comparison with a linear output activation function, and chosen for the final hyperparameter search due to its superior performance on the validation dataset. As the sigmoid outputs values in $(0, 1)$, this choice guarantees that the model will not predict nonsense values of the `soc`. After these choices were made, the hyperparameter searchspace was reduced and re-optimized for the two model architectures. The reduced search space is displayed in Table 4.1. When further optimizing the models, the number of epochs were increased to 20 000 and the



(a) RNN-oFF **(b)** RNN-oREC

Figure 4.3 Optimal architectures of the class of RNN models explored. The RNN-oFF model has 30 818 parameters and the RNN-oREC model has 8 378 parameters. The layers marked as "Dense" symbolize standard FF layers. "Dense" is the internal name of an FF layer in the Tensorflow Keras API implementation.

minimum improvement criterion for the early stopping algorithm was set to zero along with a patience of 100 epochs.

The optimizer used for learning was the Adam optimizer which is expected to perform well for the non-convex optimization task of fitting the parameters of the neural network[Kingma and Ba, 2014]. This has also been shown when applied to training RNN models[Hewamalage et al., 2021]. The final choices of hyperparameters for RNN-oFF and RNN-oREC models are presented in Table 4.2.

XGBoost hyperparameters

The hyperparameters for the XGBoost model also required some tuning. This was however a less cumbersome task as the architecture of the model was fixed. The initial random search for hyperparameters is shown in Section 4.5. The DART booster showed to outperform the GBTree booster and a second iteration was performed with only this boosting algorithm.

Further investigation into the importance of different hyperparameters showed

Hyperparameter search for RNN models	
Parameter	*Value*
Batch size	512, 1024, 2048
Learning rate	0.00001, 0.0001, 0.0005, 0.001
FF units	128, 256, 512
FF multiplier	4, 8, 16
LSTM units	4, 8, 16
Bidirectional	True, False
Kernel regularization	L1, L2, None
Regularization coefficient	10^{-6}, 10^{-5}
Dropout rate	0.0, 0.1, 0.2, 0.4

Table 4.1 Hyperparameters used in random search for RNN-oFF and RNN-oREC. Note that FF multiplier refers to the multiple of 49 used for the FF layer in the RNN-oREC model, whereas FF units refers to the number of nodes in the shared FF layer in the RNN-oFF model.

Final hyperparameters for RNN models		
	RNN-oFF	RNN-oREC
Parameter	*Value*	*Value*
Batch size	512	1024
Learning rate	0.0001	0.0001
LSTM units	4	4
FF units	256	784
Bidirectional	True	False
Weight decay regularization	None	L2
Regularization coefficient	0	10^{-6}
Dropout rate	0.1	0.1

Table 4.2 Final hyperparameters for the two RNN models. Note that each model is used for both the regression task and classification task (i.e. performs joint forecasts).

a discrepancy between the ones contributing to the performance on the regression task versus the classification task. Thus, it was decided to split the configurations for the two tasks as single sets of models were already generated for each target variable. The final hyperparameter configurations for the two tasks are displayed in Table 4.4.

4.6 Setup for model training

The training and optimization of the RNN models and the XGBoost models required both computational power and an extended performance tracking of configurations.

Initial hyperparameter search for XGBoost	
Parameter	*Value*
Number of estimators	25, 50, 100, 200, 400
Learning rate	0.01, 0.1, 0.2, 0.3
Gamma (min improvement)	0, 10^{-6}, 10^{-5}
Max depth	2, 6, 10
Booster	GBTree, DART
Column sample by tree	0.5, 0.75, 1
Subsample of data	0.5, 0.75, 1
Drop rate	0.1, 0.2, 0.3
Only day	True, False

Table 4.3 Hyperparameters used in random search for XGBoost models.

Final hyperparameters for XGBoost		
	Regressors	Classifiers
Parameter	*Value*	*Value*
Number of estimators	400	200
Learning rate	0.1	0.3
Gamma (min improvement)	0	0.0001
Max depth	6	6
Booster	DART	DART
Column sample by tree	0.75	0.75
Subsample of data	0.5	0.5
Drop rate	0.1	0.1
Only day	True	False

Table 4.4 Final hyperparameters for the XGBoost models for the two tasks. Note that this is a single model as opposed to the RNN models.

Model performance tracking

Since a large number of models were required to be configured, trained and evaluated, it quickly became cumbersome to track the performance of each model individually. To structure the process of finding the best performing architecture and hyperparameter configuration, the platform Weights & Biases was used to run the hyperparameter optimization and track the performance of the models.[‡]

Cloud computing cluster for model training

The models described in the report (except for the naive baseline model) were trained on a cloud computing cluster supplied by the Swedish National Institute for Computing (SNIC). A resource called Alvis, dedicated towards Artificial Intelligence research was utilized for this training. The Alvis cluster contains a large

[‡]More information about Weights & Biases can be found here: `https://wandb.ai/site`.

number of computing nodes with high performance Graphical Processing Units (GPUs) capable of handling the heavy computations needed to train the models. The compute nodes NVIDIA Tesla A40 with 48GB VRAM were used for model training.[§]

Replicability of results

In all models trained, a random seed has been set in order to achieve better replicability of the results. Although, since the models are trained on a cloud computing cluster with GPUs performing parallel computations, there will be some randomness involved in how the computations are synchronized. This GPU-induced randomness is difficult to remove without sacrificing the effectiveness of the parallel processing[Keras, 2022]. However, when performing the training of the models, the results proved to be almost fully replicable.

4.7 Performance evaluation

The metrics used for evaluating the forecasting models where different for the two different tasks and evaluated over all of the horizons.

For the regression task of predicting the SOC of the EV, the Root Mean Squared Error (RMSE) was used since it has the same unit as the original values (as opposed to the MSE, which gives the unit squared).

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

where the N is the number of observations in the dataset. Additionally, the Mean Absolute Error (MAE) was used, which is defined as

$$\text{MAE} = \frac{1}{N}\sum_{i=1}^{N}|\hat{y}_i - y_i|$$

As for the binary classification task of predicting when the EV was located at home, the metrics accuracy and F1-score were used. The accuracy is defined as

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{N}$$

where TP is the number of true positives and TN is the number of true negative classifications for the dataset. The accuracy is an easily interpretable metric, but fails to capture overly optimistic model predictions in the case of class imbalances. On the other hand, the F1-score considers both the ability to classify correctly and

[§]More information about the SNIC Alvis cloud computing service can be found here: `https://www.snic.se/resources/compute-resources/alvis/`.

the missclassifications as the harmonic mean of precision and recall. The F1-score is defined as

$$\text{F1-score} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} = \frac{\text{TP}}{\text{TP} + \frac{1}{2}(\text{FP} + \text{FN})}$$

where TP and TN are the number of true positives and negatives, and FP and FN are the number of false positives and negatives. The recall is a measure of the sensitivity of the classifier, i.e. its ability to detect a positive class. The precision is a measure of the missclassification of the classifier.

5

Results

This chapter presents the performance of four models: the naive baseline, XGBoost, RNN-oFF and RNN-oREC. The two RNN models are fundamentally different in how they output the forecasts, as shown in the architecture of the models in Figure 4.3. The chapter presents the performance of the four models in the two different market settings: FCR 1-D and FCR 2-D. The performance is shown first in general summarizing metrics and then as actual forecasts sampled from the test dataset. Lastly, the performance on each of the EVs is presented.

5.1 General performance

FCR 1-D market performance

The resulting performance metrics from using the models to forecast the EV charging behavior in the 1-D FCR market setting are shown in Table 5.2 for the test data and in Table 5.1 for the validation data. Figures displaying how the performance differs for various forecasted horizons is shown in Figure 5.1 for RMSE and Figure 5.2 for accuracy. The performance for MAE and F1-score is in line with RMSE and accuracy respectively and these graphs are thus omitted.

The performance metrics, both in terms of the mean over all forecasted time steps and per forecasted time step, show the XGBoost model to outperform the RNN models and the naive baseline. The performance of the RNN models is closer to the XGBoost model for the regression task than for the classification task, which

Model	*RMSE*	*MAE*	*Accuracy*	*F1-score*
Naive baseline	0.2566	0.1927	0.6964	0.7326
RNN-oFF	0.1556	0.1180	0.7929	0.8273
RNN-oREC	0.1580	0.1201	0.7945	0.8273
XGBoost	**0.1516**	**0.1131**	**0.8022**	**0.8333**

Table 5.1 Performance metrics on the validation data of the FCR 1-D market. Performance calculated as the mean over all forecasted time steps. Best performance is marked in bold.

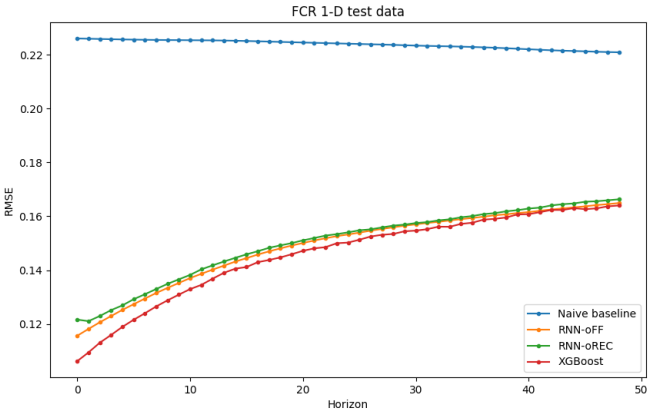


Figure 5.1 Mean RMSE per horizon for each model on the test dataset for the FCR 1-D market.

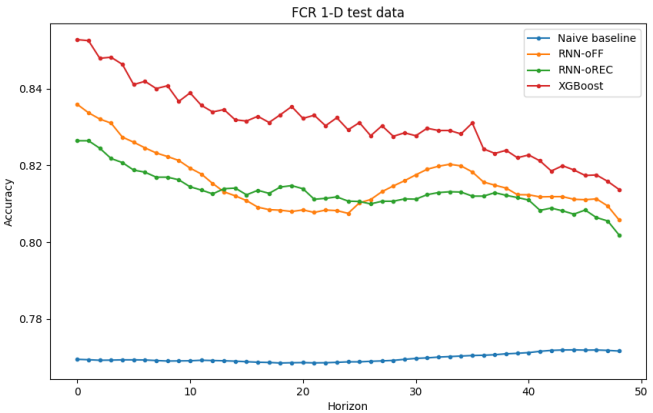


Figure 5.2 Mean accuracy per horizon for each model on the test dataset for the FCR 1-D market. The actual output of the model is the predicted probability of the EV being at the home location, and the classification is made on whether or not the predicted probability is above some threshold. For all models and horizons, the simple threshold of 0.5 was used, however this may be adjusted, yielding a trade-off between misclassification more for either of the two classes.

Model	*RMSE*	*MAE*	*Accuracy*	*F1-score*
Naive baseline	0.2239	0.1700	0.7698	0.8144
RNN-oFF	0.1488	0.1095	0.8158	0.8584
RNN-oREC	0.1501	0.1103	0.8132	0.8558
XGBoost	**0.1458**	**0.1054**	**0.8311**	**0.8692**

Table 5.2 Performance metrics on the test data of the FCR 1-D market. Performance calculated as the mean over all forecasted time steps. Best performance is marked in bold.

is most clearly shown in Figure 5.2. It should also be noted that the RNN-oFF model is superior to the RNN-oREC model on all evaluated general metrics, except for accuracy as shown in Figure 5.2.

FCR 2-D market performance

The resulting performance metrics from the models in the 2-D FCR market setting are shown in Table 5.4 for the test data and in Table 5.3 for the validation data. Figures displaying performance on different forecasted horizons are shown in Figure 5.1 for RMSE and Figure 5.2 for accuracy.

Model	*RMSE*	*MAE*	*Accuracy*	*F1-score*
Naive baseline	0.2580	0.1943	0.6981	0.7350
RNN-oFF	0.1828	0.1454	0.7457	0.7908
RNN-oREC	0.1855	0.1485	0.7396	0.7868
XGBoost	**0.1783**	**0.1418**	**0.7596**	**0.8017**

Table 5.3 Performance metrics on the validation data of the FCR 2-D market. Performance calculated as the mean over all forecasted time steps. Best performance is marked in bold.

Model	*RMSE*	*MAE*	*Accuracy*	*F1-score*
Naive baseline	0.2228	0.1689	0.7696	0.8146
RNN-oFF	0.1685	0.1320	0.7953	0.8451
RNN-oREC	0.1700	0.1347	0.7857	0.8390
XGBoost	**0.1681**	**0.1317**	**0.8088**	**0.8536**

Table 5.4 Performance metrics on the test data of the FCR 2-D market. Performance calculated as the mean over all forecasted time steps. Best performance is marked in bold.

Examining the performance of the models on the 2-D FCR market show similar ordering of the models performance-wise, although, the best performing model is not quite as clear as in the 1-D FCR market setting. The RMSE is very close for XGBoost and RNN-oFF (0.1681 vs 0.1685) on the test data, and the accuracy on the furthest horizons coincide somewhat for the two models as seen in Figure 5.4. Still, the best performing model in the FCR 2-D market is XGBoost from the general metrics.

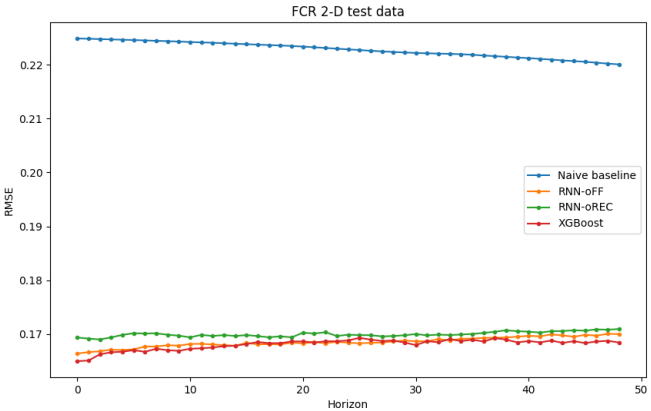


Figure 5.3 Mean RMSE per horizon for each model on the test dataset for the FCR 2-D market.

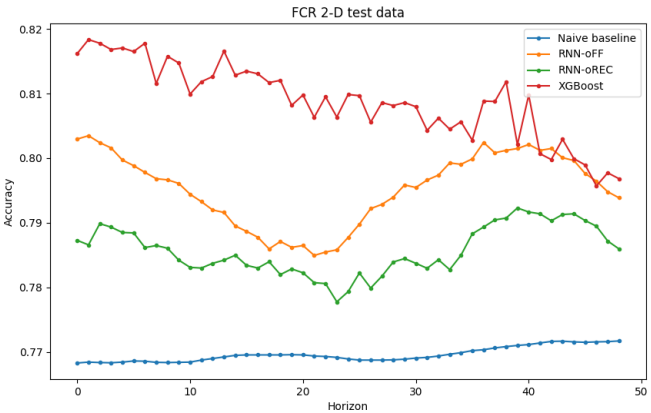


Figure 5.4 Mean accuracy per horizon for each model on the test dataset for the FCR 2-D market. The actual output of the model is the predicted probability of the EV being at the home location, and the classification is made on whether or not the predicted probability is above some threshold. For all models and horizons, the simple threshold of 0.5 was used, however this may be adjusted, yielding a trade-off between misclassification more for either of the two classes.

5.2 Example forecasts

Figure 5.5 shows an example of a generated forecast for the 1-D market for each of the four evaluated models. The example forecast is of an arbitrary EV. The same forecasts are exemplified in Figure 5.6 but for the 2-D market.

The high dimensionality of the output of all evaluated models makes it difficult to show examples on an aggregate level and still maintain a good overview, hence single examples are shown.

5.3 Performance on individual EVs

The models built are generalized models of all EVs, and the performance has thus far been evaluated on the aggregated dataset where the observations are taken from all EVs. In the setting where the model is applied, the predictions are performed on single EVs. Hence, metrics with respect to individual EVs are presented in the following figures. Figure 5.7a shows the RMSE over all EVs for the four models, and Figure 5.7b shows the same plot for the accuracy metric. The graphs show that the performance varies across EVs. Summarizing statistics were also calculated as the mean over EVs and this was shown to be in line with the summarizing statistic presented in Section 5.1.

The number of observations and the performance differs between EVs. The distribution of the number of observations and how well the models are able to forecast the charging behavior does seem to be unrelated. Hence, the summarizing statistics are not affected by the varying number of observations amongst EVs.

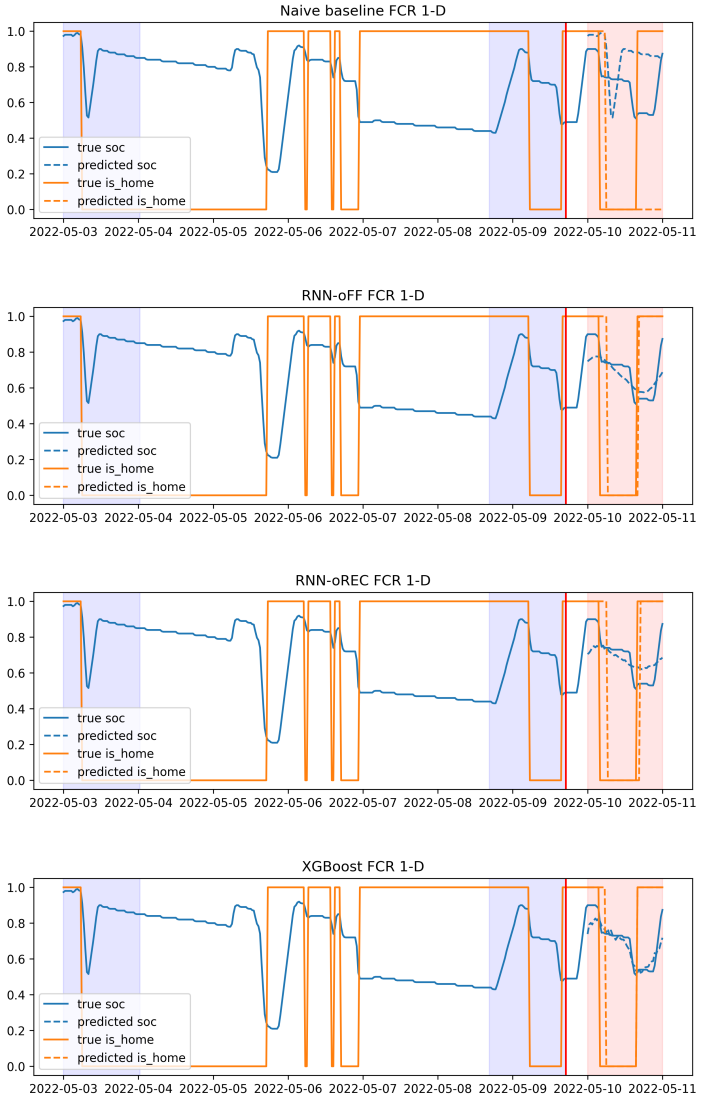


Figure 5.5 Example of forecasts for 1-D market. The example was generated for an arbitrary EV. The red vertical line signifies the forecast creation time on the 9th of May at 17:00, and the forecasted period stretches from the 10th of May at 00:00 until the 11th of May 00:00. The forecasted period is highlighted with a red marker, and the time period that constitutes the input sequences are highlighted with blue, one sequence is the 24 hours leading up to the forecast creation time and the other one is placed one week prior to the forecasted period.

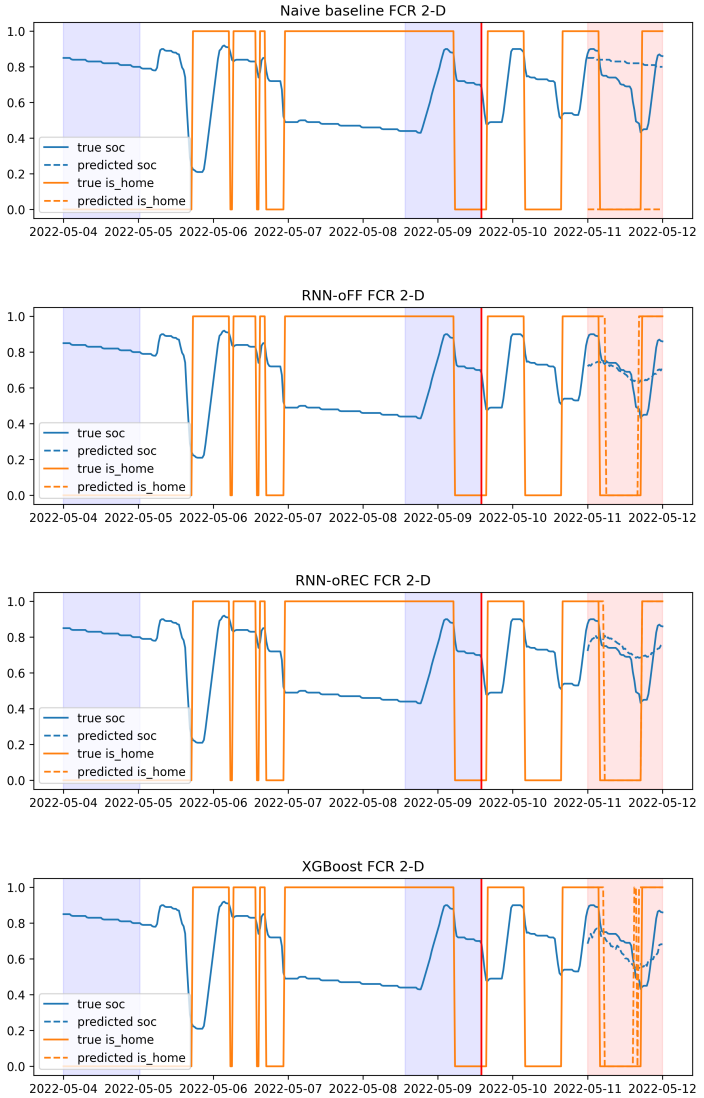
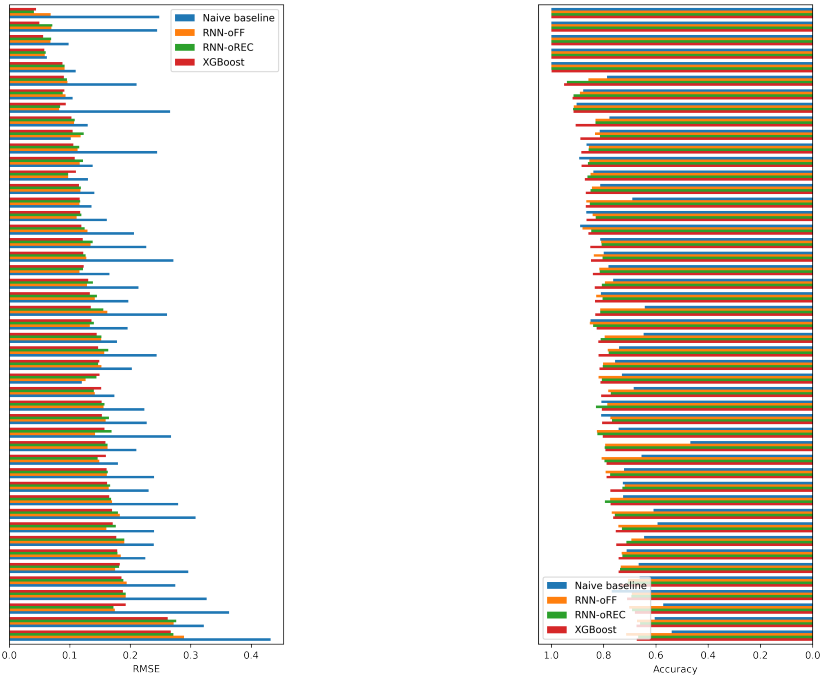


Figure 5.6 Example of forecasts for 2-D market. The example was generated for an arbitrary EV. The red vertical line signifies the forecast creation time on the 9th of May at 14:00, and the forecasted period stretches from the 11th of May at 00:00 until the 12th of May 00:00. The forecasted period is highlighted with a red marker, and the time period that constitutes the input sequences are highlighted with blue, one sequence is the 24 hours leading up to the forecast creation time and the other one is placed one week prior to the forecasted period.



(a) RMSE **(b)** Accuracy

Figure 5.7 Performance metrics per EV for the 47 vehicles on the test dataset of the FCR 1-D market. The EVs have been anonymized by removing the unique identifier. The ordering of the EVs is according to the best model (XGBoost) and is not the same for the two metrics.

6

Discussion

6.1 Performance of models

The performance of the different models are discussed and connected back to how they should be evaluated in the context of the original problem of aggregate frequency regulation on the FCR markets.

Best performance

Overall, the three statistical models applied performed well and the naive baseline was consistently beaten by the RNN models as well as the XGBoost model. The clear winner in terms of forecasting performance was the XGBoost model. The performance did not deteriorate a lot when moving from the 1-D FCR market to the 2-D FCR market (0.1458 vs. 0.1681 RMSE and 83.11 vs. 75.96 percent accuracy for the XGBoost model on 1-D and 2-D markets). These are promising results, especially as the model hyperparameters and architectures were only optimized towards the 1-D FCR market, and better results might be achievable with additional work towards the 2-D market.

Performance on test and validation datasets

Interestingly, the performance on the test data was better than on the validation data for all metrics and all models. The difference in performance was fairly large, with a 3.8 percent (0.1516 to 0.1458) RMSE improvement and a 3.4 percent (80.22 to 83.11 percent) accuracy improvement on the 1-D FCR market for XGBoost. Even larger relative and absolute performance improvements were achieved on the 2-D market. The cause of the varying performance on the two datasets could have many explanations. One hypothesis is that the Easter holidays affected the charging behavior significantly and that this period was mainly captured by the observations in the validation dataset. The distribution of time points for the observations in each dataset are displayed in Figure 4.2, and shows a clear overlap between the validation dataset and the Easter holidays (April 14th to 18th).

As for the baseline model, its performance also followed the same improvement of the other models when applied to the test data compared to the validation data.

The baseline model performed quite well overall and by all performance metrics it remained in a similar domain to the more advanced models, indicating it was not chosen to be too naive.

Performance on 1-D and 2-D FCR markets

When evaluating the performance of the models on each forecasted time step of the 1-D vs. 2-D FCR markets, some interesting patterns can be detected. For the RMSE measure on the 1-D market the three advanced models are able to perform quite well on the earliest forecasted time points, and then the performance deteriorates. This is not the case for the 2-D market where the error is stable (although worse) for all forecasted time steps. As for the accuracy metric, the RNN models show signs of sinusoidal prediction errors over the forecasted time steps, where the patterns are most clear for the RNN-oFF model. These sinusoidal patterns might be explained by the fact that the forecasting period spans 24 hours. The first horizons are the closest to the forecast creation time and thus the easiest to predict. Due to daily periodicity in the data, the behavior at the end of the forecasting horizon is similar to that of the earlier horizons. However, this sinusoidal tendency is not as apparent for the RNN-oREC model and not at all for the XGBoost model.

RNN architectures

The architectures of the two RNN models may discussed as well, since the two models are fairly similar, but as shown in Figure 4.3, the architectures differs at some points. The RNN-oFF model consistently outperforms the RNN-oREC model across all summarizing metrics in results tables, and also across time steps except for some accuracy measures on the 1-D FCR market. The RNN-oFF model has a higher model complexity and capacity with 30 818 parameters while the RNN-oREC model has 8 378 parameters. This increased capacity thus seems necessary to achieve the additional performance when forecasting.

Forecasting performance per EV

The performance of each model with respect to individual EVs was presented in Section 5.3. The ability to produce reliable forecasts differed between EVs, although, no model outperformed any other when looking at all vehicles. This insight into how the model performs with respect to different EVs is important as it helps the user of the forecasting tool to decide how to aggregate the vehicles and the uncertainty in the aggregated measurements.

Examining the accuracy measures per EV, some seemed to be unreasonably great and demanded some further investigation into the original time series of the EVs. It then occurred that one user must have moved from their home to another location and that two other users must have placed their charging stations further from the home than what was accounted for during preprocessing. These finding do surely affect the performance of the models. Recalculating the performance metrics

when omitting these three EVs does however show the overall performance to be almost identical. The forecasting performance of the models could potentially be improved by re-iterating the preprocessing of the data and training new models with higher quality data.

6.2 Limitations and improvements

Data

The data was gathered from 2022-01-04 until 2022-05-16, and was thus restricted to the late winter and spring. Even though this yielded a considerable number of observations (145 763 and 143 519 for the 1-D and 2-D FCR markets respectively), the fact that the data does not span even a single full year could influence the results in a negative way. It can be reasoned that the EV charging behavior changes over the year, especially in a country like Sweden, where the seasons affect the weather a lot. As the solution provider gathers data continuously, this issue should be resolved over time by re-training the model with new data.

The validation dataset had many observations occurring during Easter, which was hypothesized to negatively influence the performance of all models. This problem could be tackled by feeding the models information about current and upcoming holidays, as they reasonably affect the charging behavior of the users. As holidays occur throughout the year, this is a potential improvement for future research.

The choice was made to keep the 30 minute sampled data, as opposed to downsampling to e.g. 60 minute sampled data. A downsampled dataset would mean fewer inputs and outputs of all involved models, thus potentially reducing model complexity and risk of overfitting and perhaps improve the potential scalability of the forecasting models. At the same time, keeping the higher sampling frequency produces a larger number of observations, gives higher resolution in the forecasts, and a more detailed description of the predicted charging behavior which is beneficial to inform decisions in the application setting.

The choice was made to interpolate gaps of a maximum of five time steps. This was based on a trade-off between loss of usable data and judged risk of errors. Figure 2.2 illustrates the usable data as a consequence of maximum gap choice, but further interpolation may be possible without worsening the quality of the data, and could then help improve the models.

As shown in Figure 4.2, the dates overlap between the training, validation and test datasets. The implication then becomes that some predictions on the validation and training datasets take place in a point in time that the model has already seen in training, which is problematic. Each observation in the datasets is uniquely identified by the time point and EV it was gathered from, thus these overlapping time points are at least taken from different EVs, and not identical. Nevertheless, the gain of the splitting procedure used in this thesis is that each EV is represented in

all three datasets proportionally to the total number of usable observations, and thus the performance on each EV should be better represented.

The decision regarding what data to feed the models was that the sliding window described in Section 4.3 allowed for overlap. The use-case for the models is for the 1-D models to make predictions only at 17:00 and for the 2-D models to only make predictions at 14:00. It can be reasoned that the models should only be trained on the exact task they are expected to perform. Although, had overlap not been applied, the amount of usable data would decrease by a factor of 48.

Models

A conscious choice was made to model the charging behavior of a generalized EV. The decision was made without fundamental insight about the individual EVs' charging behavior, which may make the assumption that a generalized behavior exists flawed. However, to highlight differences in charging behavior a model of said behavior is required, which was the goal of the thesis to begin with. This thesis may lay the ground work for a more sophisticated choice of data selection and clustering for the future.

The forecasting problem of the thesis regards two fundamentally different predictions: the regression task and the classification task. For the XGBoost model, this was separated as the two tasks could not be performed jointly in the XGBoost framework, with fundamentally different models and training procedures. This in combination with the relatively small variations in architecture and hyperparameters compared to the RNN models meant it was feasible to optimize the XGBoost model configuration for the two tasks separately.

For the RNN models, the modelling was performed jointly, i.e. the same network was trained to approach both the classification and regression tasks simultaneously. This was a pragmatic choice, and initial exploring showed that the models' performance was not affected by this choice. In any case, this could be seen as giving the XGBoost a more beneficial set of circumstances to perform in, and might contribute to the superior performance of this model.

For all evaluated models, no optimization was done on the separate horizons, again a pragmatic decision. This too, could change the performance of the models used. Furthermore, the optimization of hyperparameters and architecture was conducted solely on the 1-D market setting and then directly applied to the 2-D setting. Repeating the procedure for the 2-D setting might have improved the results.

For the RNN models, the output activation functions may be chosen with some freedom. For the binary classification, the sigmoid was applied as it is common practice when dealing with such problems. As for the `soc`, the choice was not so obvious. For regression tasks in a neural network setting, a linear activation function may perform well, but one key characteristic of this problem is that the `soc` is

bounded on the interval $[0, 1]$. Thus, the sigmoid and linear activation functions were tried, and the sigmoid proved to give more satisfactory results. The choice makes sense as it guarantees that the forecasted values of `soc` will never be some nonsense value outside the interval $[0, 1]$. However, the choice might mean that the predictions of `soc` are smoothed out and rarely assume values close to 0 or to 1. Figure 5.5 and Figure 5.6 show that such a tendency might be apparent.

The choice of input window was presented and motivated in Section 4.3, and was kept constant throughout the project. It is possible that different choices of input sequences might improve the overall performance. From the XGBoost models constructed, the was an indication that the input data from one week prior was a better predictor of the EV being located at home, while the input data from 24 hours prior to the forecast creation time was a better predictor of the SOC.

Furthermore the feature selection could be performed differently. The results in this thesis were all found by inputting all 15 features presented in Table 2.2. Attempts were made during experimentation to reduce the number of features seen by the models, however with unsatisfactory results. Thus they were all kept, as a pragmatic consideration. A final remark about the limitations on the models may be made that out of the many variations of RNN, only the LSTM unit was used. The choice was based on the success that LSTMs have had on time series modelling and their robustness to long time dependencies.

Applicability

As the application of the forecasts is quite involved and connected to the solution provider's proprietary control algorithm, its details have not been pursued in this thesis. The main quantity which should be forecasted is in reality the amount of electrical load which may be shifted through the solution provider's scheduling of EV charging. Translating the two forecasted time series into a measure of flexibility may be performed through three quantities: (1) time of plug in, (2) time of plug out, and (3) state of charge at the time of plug in. Hence, an improvement to the models applied may be to further optimize towards the exact quantities stated.

The measure of flexibility should also be considered on an aggregate level when bidding on the FCR markets where the forecasts are utilized. Errors on individual EVs could thus be cancelled by errors in the opposite direction for other EVs on an aggregate level. Thus, the aggregate flexibility measure should be more reliable than the individual one.

It should also be noted that the binary classification output of the models are probabilities for the EV being located at home (except for the baseline model which outputs the predicted *True* or *False*). With the probabilities at hand, the solution provider may choose some threshold based on the desired amount of risk.

For the end user of the models explored in the thesis, there are some final practical considerations. When moving into deployment of the model, the full dataset can

be used in order to make sure that the model generalizes in the best possible way. Furthermore, since the data used for modeling is only representative of late winter and spring, as more data is gathered, the model would have to be re-trained in order to ensure consistent performance.

7

Conclusion

The aim of this thesis was to create reliable forecasts of EV charging behavior, which in turn can be applied by the solution provider to provide frequency regulation through the FCR markets. Two different problem settings exist for forecasting: the 1-D and 2-D markets. One XGBoost model and two variations of RNN models were applied and compared to a naive baseline.

The forecasting problem consists of two fundamentally different tasks: predicting if the EV is at the home location or not (classification), and the state of charge of the EV (regression). The performance of the models on the two tasks was judged by accuracy and F1-score for the classification, and by RMSE and MAE for the regression. By all metrics, all three advanced models consistently beat the baseline.

The XGBoost model also proved to out-perform all of the other evaluated models. This can be nuanced by the fact that the XGBoost model was tailored to learn the regression and classification tasks separately, which may have given it an edge on the RNN models. It may be seen as surprising that the best model for a time series forecasting task turned out to be a model where the time dependency of the data was not encoded at all. The conclusion of the thesis shows promising results of applying these type of models to forecasting of EV charging behavior.

7.1 Future Research

Various trade-offs and limitations of the thesis were discussed in the previous chapter. Throughout the project, ideas for improving results and new directions of research were discovered, and some are presented in this final section of the thesis.

Specialized loss function

The loss function for training ML models dictates what phenomena the models are trained to capture. In the thesis, the models are trained to forecast the state of charge and when the EV is at home, and the measure of flexibility may then be calculated from these quantities. Future research topics may include tailoring the training process to focus on forecasting the flexibility measure explicitly. This could

be accomplished by a more specialized loss function to help the model to learn the desired task directly.

Model architecture

Recent research has shown that incorporating attention mechanisms has successfully improved the performance of sequence modelling for RNNs by leveraging automatic feature selection. Applying more advanced attention mechanisms could potentially benefit this forecasting problem [Vaswani et al., 2017].

The XGBoost models explored in the thesis are inherently single output models and a collection of 98 individual models were required to fit the forecasting problem. Naturally, there is a strong dependence among the individual models as the outputs form a time series. Thus, the correlation may be leveraged to improve both the efficiency of training and the performance of the models. These concepts are further explored in [Zhang and Jung, 2019] and there is currently experimental support through the XGBoost implementation [XGBoost, 2022b].

Feature engineering and clustering

All of the approaches explored in this thesis could be helped by additional external data resources, such as holidays and outdoor temperature. Conducting a systematic feature selection should decrease the complexity of the models and reduce the risk of overfitting.

Furthermore, the available data for this problem grows over time, which will benefit the modelling of charging behavior as the models used have more examples to learn from. As the models train on data from different parts of the year, season can be encoded and the models can learn a wider variation of behavior.

The models presented in this thesis may serve as an initial description of EV charging behavior. By use of these models, differences in behavior among individual users can be identified, and clustering approaches may be applied.

Bibliography

Bandara, K., C. Bergmeir, and S. Smyl (2020). "Forecasting across time series databases using recurrent neural networks on groups of similar series: a clustering approach". *Expert Systems with Applications* **140**, p. 112896. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2019.112896. URL: https://www.sciencedirect.com/science/article/pii/S0957417419306128.

Bergstra, J. and Y. Bengio (2012). "Random search for hyper-parameter optimization". *Journal of Machine Learning Research* **13**:10, pp. 281–305. URL: http://jmlr.org/papers/v13/bergstra12a.html.

Bianchi, F. M., E. Maiorino, M. C. Kampffmeyer, A. Rizzi, and R. Jenssen (2017). "An overview and comparative analysis of recurrent neural networks for short term load forecasting". *CoRR* **abs/1705.04378**. arXiv: 1705.04378. URL: http://arxiv.org/abs/1705.04378.

Chen, T. and C. Guestrin (2016). "XGBoost". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM. DOI: 10.1145/2939672.2939785. URL: https://doi.org/10.1145%5C%2F2939672.2939785.

Chollet, F. (2017). *Deep Learning with Python*. 1st. Manning Publications Co., USA. ISBN: 1617294438.

Dodson, T. and S. Slater (2019). *Electric Vehicle Charging Behaviour Study*. URL: http://www.element-energy.co.uk/wordpress/wp-content/uploads/2019/04/20190329-NG-EV-CHARGING-BEHAVIOUR-STUDY-FINAL-REPORT-V1-EXTERNAL.pdf.

Emulate Energy (2022). *Emulate energy*. English. URL: https://www.emulate.energy/.

Friedman, J. H. (2001). "Greedy function approximation: a gradient boosting machine". *The Annals of Statistics* **29**:5, pp. 1189–1232. ISSN: 00905364. URL: http://www.jstor.org/stable/2699986 (visited on 2022-05-25).

Hecht, C., J. Figgener, and D. U. Sauer (2021). "Predicting electric vehicle charging station availability using ensemble machine learning". *Energies* **14**:23. ISSN: 1996-1073. DOI: 10.3390/en14237834. URL: https://www.mdpi.com/1996-1073/14/23/7834.

Hewamalage, H., C. Bergmeir, and K. Bandara (2021). "Recurrent neural networks for time series forecasting: current status and future directions". *International Journal of Forecasting* **37**:1, pp. 388–427. DOI: 10.1016/j.ijforecast.2020.06.008. URL: https://doi.org/10.1016%2Fj.ijforecast.2020.06.008.

Hochreiter, S. and J. Schmidhuber (1997). "Long short-term memory." *Neural Computation* **9**:8, pp. 1735–1780. URL: http://ludwig.lub.lu.se/login?url=https://search.ebscohost.com/login.aspx?direct=true&AuthType=ip,uid&db=inh&AN=5736179&site=eds-live&scope=site.

International Energy Agency (2022). *Energy transitions require innovation in power system planning*. URL: https://www.iea.org/articles/energy-transitions-require-innovation-in-power-system-planning.

James, G., D. Witten, T. Hastie, and R. Tibshirani (2021). *An Introduction to Statistical Learning: with Applications in R*. Springer Texts in Statistics. Springer US. ISBN: 9781071614174. DOI: 10.1007/978-1-0716-1418-1. URL: https://link.springer.com/10.1007/978-1-0716-1418-1.

Jozefowicz, R., W. Zaremba, and I. Sutskever (2015). "An empirical exploration of recurrent network architectures". In: *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*. ICML'15. JMLR.org, Lille, France, pp. 2342–2350.

Keras (2022). *How can i obtain reproducible results using keras during development?* URL: https://keras.io/getting_started/faq/#how-can-i-obtain-reproducible-results-using-keras-during-development.

Kingma, D. P. and J. Ba (2014). *Adam: a method for stochastic optimization*. DOI: 10.48550/ARXIV.1412.6980. URL: https://arxiv.org/abs/1412.6980.

Lim, B. and S. Zohren (2021). "Time-series forecasting with deep learning: a survey". *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* **379**:2194, p. 20200209. DOI: 10.1098/rsta.2020.0209. URL: https://doi.org/10.1098%2Frsta.2020.0209.

Madjidian, D., M. Roozbehani, and M. A. Dahleh (2018). "Energy storage from aggregate deferrable demand: fundamental trade-offs and scheduling policies". *IEEE Transactions on Power Systems* **33**:4, pp. 3573–3586. DOI: 10.1109/TPWRS.2017.2766144.

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Y. Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng (2015). *TensorFlow: large-scale machine learning on heterogeneous systems*. Software available from tensorflow.org. URL: https://www.tensorflow.org/.

Olah, C. (2015). *Understanding lstm networks – colah's blog*. English. URL: https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

Rashmi, K. V. and R. Gilad-Bachrach (2015). "Dart: dropouts meet multiple additive regression trees." URL: http://ludwig.lub.lu.se/login?url=https://search.ebscohost.com/login.aspx?direct=true&AuthType=ip,uid&db=edsarx&AN=edsarx.1505.01866&site=eds-live&scope=site.

Schuster, M. and K. Paliwal (1997). "Bidirectional recurrent neural networks". *IEEE Transactions on Signal Processing* **45**:11, pp. 2673–2681. DOI: 10.1109/78.650093.

Sharma, S., S. Sharma, and A. Athaiya (2020). "Activation functions in neural networks". In:

Shifaz, A., C. Pelletier, F. Petitjean, and G. I. Webb (2021). *Elastic similarity measures for multivariate time series classification*. DOI: 10.48550/ARXIV.2102.10231. URL: https://arxiv.org/abs/2102.10231.

Svenska Kraftnät (2022a). *Bilaga 3: villkor för fcr, avtal 4620-3*. Swedish. URL: https://www.svk.se/siteassets/4.aktorsportalen/systemdrift-o-elmarknad/balansansvar/aktuella-balansansvarsavtal/4620_3-bilaga-3-fcr.pdf.

Svenska Kraftnät (2022b). *Frekvenshållningsreserv normaldrift (fcr-n)*. Swedish. URL: https://www.svk.se/aktorsportalen/systemdrift-elmarknad/information-om-stodtjanster/fcr-n/.

Svenska Kraftnät (2022c). *Guidance on the provision of reserves*. English. URL: https://www.svk.se/siteassets/4.aktorsportalen/systemdrift-o-elmarknad/information-om-stodtjanster/guidance-on-the-provision-of-reserves.pdf.

Svenska Kraftnät (2022d). *Svenska kraftnät börjar upphandla fcr-d ned för 2022*. Swedish. URL: https://www.svk.se/press-och-nyheter/nyheter/elmarknad-allmant/2021/svenska-kraftnat-borjar-upphandla-fcr-d-ned-for-2022/.

Taieb, S. B., G. Bontempi, A. Atiya, and A. Sorjamaa (2011). *A review and comparison of strategies for multi-step ahead time series forecasting based on the nn5 forecasting competition*. DOI: 10.48550/ARXIV.1108.3259. URL: https://arxiv.org/abs/1108.3259.

Ullah, I., K. Liu, T. Yamamoto, R. E. A. Mamlook, and A. Jamal (2021a). "A comparative performance of machine learning algorithm to predict electric vehicles energy consumption: a path towards sustainability". *Energy & Environment* **0**:0, p. 0958305X211044998. DOI: 10.1177/0958305X211044998. eprint: https://doi.org/10.1177/0958305X211044998. URL: https://doi.org/10.1177/0958305X211044998.

Ullah, I., K. Liu, T. Yamamoto, M. Zahid, and A. Jamal (2021b). "Electric vehicle energy consumption prediction using stacked generalization: an ensemble learning approach". *International Journal of Green Energy* **18**:9, pp. 896–909. DOI: 10.1080/15435075.2021.1881902. eprint: https://doi.org/10.1080/15435075.2021.1881902. URL: https://doi.org/10.1080/15435075.2021.1881902.

Vaswani, A., N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin (2017). *Attention is all you need*. DOI: 10.48550/ARXIV.1706.03762. URL: https://arxiv.org/abs/1706.03762.

Wen, R., K. Torkkola, B. Narayanaswamy, and D. Madeka (2017). *A multi-horizon quantile recurrent forecaster*. DOI: 10.48550/ARXIV.1711.11053. URL: https://arxiv.org/abs/1711.11053.

XGBoost (2022a). *Extreme gradient boosting*. URL: https://github.com/dmlc/xgboost.

XGBoost (2022b). *Xgboost multiple outputs*. URL: https://xgboost.readthedocs.io/en/stable/tutorials/multioutput.html.

Yang, X., C. Chen, W. Zhao, and Y. Li (2021). "Electric vehicle load forecasting in distribution transformer based on feature engineering". In: *2021 IEEE 4th International Electrical and Energy Conference (CIEEC)*, pp. 1–5. DOI: 10.1109/CIEEC50170.2021.9510549.

Zhang, Z. and C. Jung (2019). *Gbdt-mo: gradient boosted decision trees for multiple outputs*. DOI: 10.48550/ARXIV.1909.04373. URL: https://arxiv.org/abs/1909.04373.

A

Data attributes

A.1 EV time series attributes

1. `time` - Datatype: String. Description: encodes date and time of the measurement using the TZ-format ("YYYY-MM-DDThh:mm:ss.fffZ")

2. `period` - Datatype: Integer. Description: Counting sequence of number of minutes passed since the EV was onboarded, i.e. has increments of 30, the sampling frequency.

3. `soc` - Datatype: Float. Description: State of charge of the EV battery in decimal form, i.e. `soc` $\in [0, 1]$ at all time points (0 % to 100 %).

4. `odometer` - Datatype: Float. Description: Measurements from the cars internal odometer, measuring the total distance traveled by the EV.

5. `latitude` - Datatype: Float. Description: Measurements of the latitude at which the EV is located.

6. `longitude` - Datatype: Float. Description: Measurements of the longitude at which the EV is located.

7. `charge_status` - Datatype: String/Categorical. Description: Indicates Charge status of the EV, which falls into one out of three categories, `charge_status` $\in \{$`CHARGING`, `NOT_CHARGING`, `FULLY_CHARGED`$\}$

8. `isPluggedIn` - Datatype: Boolean/Categorical. Description: Indicates whether or not the EV is plugged in to a charging station. `isPluggedIn` $\in \{$True, False$\}$

A.2 EV metadata attributes

1. `id` - Datatype: Integer. Description: Unique identifier of specific EV.

2. `state` - Datatype: String/Categorical. Description: Indicates if the EV is actively participating in the data gathering process, `state` $\in$ {`active`, `pending`}.

3. `manufacturer` - Datatype: String. Description: Indicates the manufacturer of the EV.

4. `product` - Datatype: String. Description: Indicates the name of the product.

5. `customer` - Datatype: Integer. Description: Identification number of the customer that owns the EV.

6. `onboarded_at` - Datatype: String. Description: Date and time of onboarding of the EV, i.e. date and time that data started being collected. Encoded using the TZ-format ("YYYY-MM-DDThh:mm:ss.fffZ")

7. `home_location` - Datatype: Tuple of floats. Description: Gives coordinates of home location for the EV on the format "(latitude, longitude)".

8. `battery_capacity` - Datatype: Float. Indicates the preferred state of charge of the EV, i.e. the state of charge at which it is considered fully charged. This is not necessarily 100 %, but is a setting that may be tweaked by the customers at any time.

B

Model parameters

B.1 RNN hyperparameters

Parameter	*Description*
FF units	# of nodes in a FF layer
LSTM-units	# of LSTM cells in a layer
Out activation function, soc	Final activation function in output layer for soc forecasts
Optimizer	Algorithm used in training
Batch size	Number of observations in randomly selected batch
Epochs	Number of epochs in training
Learning rate	Size of steps in negative gradient direction
Min delta	Minimum improvement between epochs in measured loss
Patience	Number of epochs that training can violate min delta before early stopping
Kernel regularization	What penalty is put on large weights, L1 or L2 norm
Regularization coefficient	Coefficient before the penalty on weight size
Dropout rate, FF layer	Probability of killing weights connected to FF layer
Dropout rate, LSTM layer	Probability of killing weights connected to LSTM layer

Table B.1 Explanation of hyperparameters for RNN models

B.2 XGBoost hyperparameters

Parameter	*Description*
Number of estimators	Number of trees built by model
Learning rate	How quickly model should fit the training data
Gamma	Minimum improvement for tree to be added
Max depth	Maximum depth of each tree
Booster	Boosting technique, either DART or GBTree
Column sample by tree	Percentage of how columns shown to each tree
Subsample of data	Percentage of how many observations shown to each tree
Drop rate	Probability of dropping tree (only when DART used as booster)
Only day	If true, only yesterday-sequence is used, otherwise last week as well

Table B.2 Explanation of hyperparameters for XGBoost models.

Lund University
Department of Automatic Control
Box 118
SE-221 00 Lund Sweden

Document name
MASTER'S THESIS

Date of issue
June 2022

Document Number
TFRT-6169

Author(s)
Fredrik Sidh
Gustaf Sundell

Supervisor
Daria Madjidian, Emulate Energy AB
Richard Pates, Dept. of Automatic Control, Lund University, Sweden
Anders Rantzer, Dept. of Automatic Control, Lund University, Sweden (examiner)

Title and subtitle
Data-driven forecasting of electric vehicle charging for frequency regulation

Abstract

Electric vehicle (EV) charging may be used in aggregation as virtual batteries to provide a frequency regulating service to the power grid. The service is sold on the Frequency Containment Reserve (FCR) markets, and is traded one and two days ahead. Forecasts of charging patterns are essential to reliably provide this ancillary service. The thesis aims to build a generalized model for forecasting EV charging behavior of 47 EVs in Sweden. The charging behavior is characterized by the state of charge and whether the EV is plugged in to the home charging station or not. Recurrent Neural Networks (RNNs) and XGBoost are applied to produce forecasts that fit the two FCR market settings. Performance of the models is evaluated and compared to a naive baseline in terms of RMSE, MAE, accuracy and F1-score. The naive baseline assumes the same charging behavior as the previous week. The results show both classes of models to consistently beat the naive baseline on both markets, and XGBoost proved to be the best forecaster.

Keywords

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title
0280-5316

ISBN

Language
English

Number of pages
1-73

Recipient's notes

Security classification