Karlsruhe Institute of Technology

# Probabilistic Forecasting of Energy Time Series using Deep Learning

Master's Thesis of

Yannick Tanner

at the Department of Informatics
Institute for Automation and Applied Informatics (IAI)

| | |
|---|---|
| Reviewer: | Prof. Dr. Veit Hagenmeyer |
| Second reviewer: | Prof. Dr. Achim Streit |
| Advisor: | Nicole Ludwig, M.Sc |

15.09.2019 – 16.03.2020

Karlsruher Institut für Technologie
Fakultät für Informatik
Postfach 6980
76128 Karlsruhe

I declare that I have developed and written the enclosed thesis completely by myself, and have not used sources or means without declaration in the text.

**Karlsruhe, 16.03.2020**

. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

(Yannick Tanner)

# Abstract

Deep Learning methods are widely successful and continue to be applied in new fields, one of them being forecasting of energy time series. However, they lack an indication of their confidence in predictions, which helps evaluate, interpret, and improve forecasters.

Therefore, this thesis explores probabilistic extensions to Deep Learning and their application in forecasting of energy time series. The methods tested are Concrete Dropout, Deep Ensembles, Bayesian Neural Networks, Deep Gaussian Processes, and Functional Neural Processes. For evaluation, two load forecasting scenarios are considered: Short-Term (single-step) and day-ahead (multi-step) forecasting. The methods are evaluated in terms of calibration, sharpness, and how well they indicate a lack of knowledge (Epistemic Uncertainty). As reference a simple Neural Network and a Quantile Regression model are used.

Overall, the methods perform well, with Concrete Dropout, Deep Ensembles, and Bayesian Neural Networks performing similarly well or better to the reference methods. Functional Neural Processes and Deep Gaussian Processes perform worse, likely due to a lack of convergence and sub-optimal parameters. Deep Ensembles in particular prove to be simple to implement and train and to use very few hyper-parameters. Concrete Dropout and Bayesian Neural Networks show similar advantages but need additional configuration for good Epistemic Uncertainty estimates. Furthermore, Concrete Dropout and Deep Ensembles are comparatively fast in training and predictions.

# Contents

# List of Figures

# List of Tables

# 1. Introduction

The last decade defines the rise of Neural Networks and the concept of Deep Learning (LeCun et al. 2015), mainly because modern computing capabilities enabled the training of deep models in a reasonable time frame (Chellapilla et al. 2006; Oh and Jung 2004). This led to the enthusiastic application of the methods in a growing amount of fields, with a lot of success in image recognition using Convolutional Neural Networks (CNNs) (Krizhevsky et al. 2012).

In theory a Neural Network can approximate any function given enough neurons (Hornik 1991; Cybenko 1989; Csáji et al. 2001). At the same time, there are methods that have the same theoretical function approximation capabilities (e.g. Support Vector Machines (SVMs) (Hammer and Gersmann 2003)). The real power of Neural Networks lies in their natural support for layering them in a deep structure. The layering allows learning intrinsic feature representations during training (i.e., higher layers can build upon the abstraction of the lower layers). Although, the representations are not perfect, highly dependent on the training data, and can be counterintuitive (Szegedy et al. 2013), they generally improve the trained model enabling it to recognize different patterns on different layers with different levels of abstraction. The success of Neural Networks has also led to research into layering different methods in a similar way into deep models (Damianou and Lawrence 2013; Zhou and Feng 2017), incorporating Neural Networks with methods of other domains, like graphs or latent variables, to create new methods (Garnelo et al. 2018b; Louizos et al. 2019; Scarselli et al. 2008), and into improving on the shortcomings of Neural Networks themselves, like generating adversarial training data (Goodfellow et al. 2014) or the probabilistic extensions discussed in this thesis.

While being broadly and successfully applied in many contexts, standard Neural Networks have a major drawback. They lack an estimate of how confident they are in their predictions (Gal and Ghahramani 2016; Seedat and Kanan 2019). A lack of uncertainty information puts the user at a disadvantage. Having uncertain output data helps in understanding how well the model learned something, and where there might be room for improvement. If the model is overconfident or not confident enough it might be viable to get more or more expressive training data. Furthermore, information on uncertainty also helps in making sound decisions based on the model output. In most fields an estimate of how confident a prediction, a classification or, an estimation is helps with the interpretation and utilization of these results. For example, if a model outputs a low confidence it may warrant a closer inspection of the result as opposed to using the result as is.

The primary goals of this thesis are to explore probabilistic extensions to Deep Learning, to apply them in the domain of energy time series forecasting, and to evaluate the results. To achieve this, probabilistic extensions to Neural Networks, namely Bayesian Neural Networks, Deep Ensembles, Functional Neural Processes and Concrete Dropout, as well as a probabilistic method that can be structured in a deep hierarchy, namely Deep Gaussian Processes are explored.

At the point of writing, only a limited amount of research in the area of probabilistic Deep Learning for forecasting has been published. Therefore, configuring and evaluating probabilistic deep models in the field of energy time series makes a contribution to the scientific progress in establishing Deep Learning approaches as standard methods and close the gap of uncertainty estimation in comparison to statistical forecasting approaches. For forecasting with energy time series uncertainty estimates are particularly interesting, as the continuing growth of renewable energy, in particular solar and wind power, introduces volatility to the energy system and more flexibility in all areas is needed. Consequently, a confidence estimate for forecasts helps make better decisions under the increasing amount of uncertainty.

The methods are evaluated for a load forecasting scenario and compared to a simple Neural Network and a Quantile Regression model as reference. The methods perform similarly well, with Concrete Dropout and Deep Ensembles showing significant advantages in terms of simplicity and training time. Functional Neural Processes and Deep Gaussian Processes tend to converge slower than the other methods and, overall, did not perform as well, but can likely be optimized further. Other than Functional Neural Processes and Deep Gaussian Processes, all methods performed similarly to the reference models. The introduced models also have the advantage of basing their confidence estimate not only in the distribution of the training data, but also in their own confidence based in similarity of a situation to the relevant training data.

The thesis is structured as follows: first we explore the foundations of probabilistic Deep Learning by investigating the uncertainty that is produced (Section 2.1) and introducing the research that forms the basis of the methods we will be working with (Section 2.2). Building on the foundations, related research into probabilistic forecasting using Deep Learning is introduced (Chapter 3). The methods are selected, refined (Chapter 4) and evaluated in a load forecasting scenario (Chapter 5). Finally we discuss the results of the evaluation (Chapter 6) and conclude the findings (Chapter 7).

# 2. Foundations

This chapter focuses on introducing the foundations for probabilistic forecasting using Deep Learning.

## 2.1. Uncertainty

In the introduction we explored why it is beneficial to have an uncertainty estimate for predictions. This section gives deeper insight into what is generally understood when considering uncertainty estimates in a Machine Learning context. There are a variety of different types of uncertainty, in the following we will explore the ones that are used in most related research.

### 2.1.1. Aleatoric Uncertainty

The Aleatoric Uncertainty is th uncertainty inherent in the observations. This uncertainty is divided into two categories:

**Homoscedastic** The uncertainty stays constant.

**Heteroscedastic** The uncertainty is different and is dependent on input data

In practice, modeling of Aleatoric Uncertainty depends on the type it is assumed to be. **Heteroscedastic** uncertainty is usually a model output, while **Homoscedastic** uncertainty can be approximated independently of the model (Kendall and Gal 2017; Kendall et al. 2018; Shridhar et al. 2018; Der Kiureghian and Ditlevsen 2009).

### 2.1.2. Epistemic Uncertainty

Epistemic Uncertainty or Model Uncertainty is often described as the uncertainty about how well a model explains the data, due to a lack of information that, in theory, could be provided (Shridhar et al. 2018; Der Kiureghian and Ditlevsen 2009).

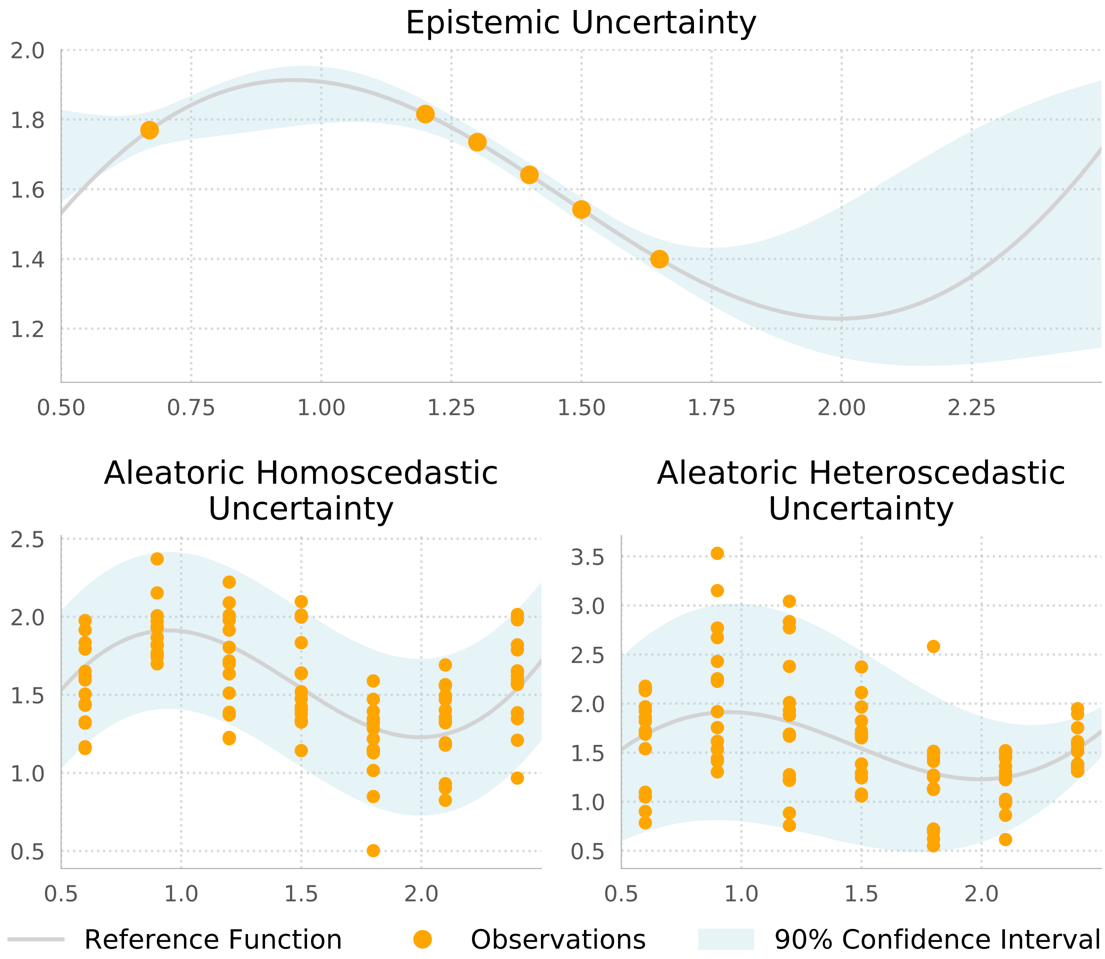


**Figure 2.1.** Visualization of the types of uncertainty in the context of a Machine Learning scenario. The top plot shows the Epistemic Uncertainty produced by a model trying to fit a reference function (gray) by learning from a set of observations (orange) (in this case the model is a Gaussian Process). In the bottom left and right the Aleatoric Homoscedastic Uncertainty and the Aleatoric Heteroscedastic Uncertainty are visualized. The uncertainty follows the uncertainty of the observations. For the Homoscedastic case the uncertainty is the same for all points while for the Heteroscedastic case the uncertainty is variable.

### 2.1.3. Importance of the Types of Uncertainty

The definition of these uncertainties does allow for some interpretation. In fact, it all depends on where the line is drawn between the information that can be theoretically provided to reduce the Epistemic Uncertainty and the information that cannot be provided. For example, Aleatoric Uncertainty is often caused by measurement noise, it could be argued that better measurements would theoretically be possible. In the context of Machine Learning, Aleatoric Uncertainty is usually the uncertainty that is inherent in the training data. Aleatoric Uncertainty represents the distribution of the data, independently of where this distribution actually comes from or the amount of data that is available. The Epistemic Uncertainty or Model Uncertainty is the trained model's confidence in the output. The confidence of the model mainly depends on the amount of training data and the amount of training. The assumption is that if there was more training data for a specific prediction that should result in a better model (i.e., a model with a higher confidence for that point) (Kendall and Gal 2017; Kendall et al. 2018; Shridhar et al. 2018; Der Kiureghian and Ditlevsen 2009).

In Figure 2.1 the types of uncertainty are visualized. For points without data a model is forced to guess. The resulting Epistemic Uncertainty is high while it is low at points where data has been observed. The Aleatoric Uncertainty represents the uncertainty inherent in the observations (i.e., the spread of the observations).

Both the described types of uncertainty are important and part of the overall uncertainty estimate. Ignoring one of them will most likely result in an underestimation of the uncertainty. Aleatoric Uncertainty gives an estimate of the distribution of the training data, it models the lack of confidence of the data source itself. Epistemic Uncertainty represents the confidence of the model, based on how well the training data explains a new situation.

## 2.2. Methods

The following section explores the foundations of some different methods that allow uncertainty estimates in Deep Learning.

A typical application of Neural Networks is classification, it should be noted that for classification problems it is quite easy to get a probabilistic output from a Neural Network. With the right loss function, activation function, and a one-hot encoding of the classes the output represents the posterior probabilities of class membership (Bishop 1994). This is not the case for prediction of continuous values.

For probability estimates in a continuous case we need to turn to other methods. Table 2.1 gives a short comparison of the methods, explored in this section.

### 2.2.1. Mixture Density Networks

A very simple probabilistic extension to classical Neural Networks are Mixture Density Networks (MDNs) (Bishop 1994). A Neural Network is trained to output a mixture of

**Table 2.1.** Short comparison of the methods introduced in foundations. Note: Even though some approaches are proposed to predict Aleatoric Uncertainty via model output and a appropriate loss function, they are not marked to be able to estimate Aleatoric Uncertainty in this table, as this approach is applicable to all methods (see Section 4.2).

| | MDNs | MC Drop. | Conc. Drop. | Deep Ens. | BNNs | Deep GPs | FNPs | NPs |
|---|---|---|---|---|---|---|---|---|
| Neural Architecture | ✓ | ✓ | ✓ | ✓ | ✓ | ✗ | ✓ | ✓ |
| Epistemic Unc. | ✗ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Aleatoric Unc. | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✓ | ✓ |
| Approx. Training Effort | + | ++ | ++ | ++ | +++ | ++++ | +++ | ++ |
| Approx. Prediction Effort | + | ++ | ++ | + | ++ | + | +++ | ++ |
| Scaling With Data | ++ | ++ | ++ | ++ | + | − | + | + |
| Application Of Prior Knowledge | − | − | − | − | + | ++ | − | ++ |
| Hyper-Parameters | + | ++ | ++ | + | ++ | ++ | ++++ | ++ |

gaussian distributions by learning a mean, a variance, and a weight for a number of distributions. The number has to be set beforehand.

MDNs were specifically motivated by the problem that the average of correct outputs, a traditional Neural Network will produce, is not bound to be a correct value itself. Specifically the example of inverse kinematics is mentioned where one input can have multiple outputs (Bishop 1994).

A similar idea is to use Quantile Loss instead of other Loss functions to train a regression model. This will result in the regression being a quantile (depending on a parameter in the loss function). Doing this multiple times results in multiple quantiles representing the distribution of the data (Koenker and Hallock 2001).

Both Quantile Regression in Neural Networks and Mixture Density Networks try to approximate the uncertainty already in the training data (Aleatoric Uncertainty). These methods will not give an estimate of the Model Uncertainty (Epistemic Uncertainty).

### 2.2.2. Monte-Carlo Dropout

A very straightforward Epistemic Uncertainty extension to Neural Networks is Monte-Carlo Dropout.

In general, Dropout in Neural Networks is a technique to avoid overfitting by randomly disabling (dropping out) units during training. This can be seen as applying random noise in training (Hinton et al. 2012; Srivastava et al. 2014).
Monte-Carlo Dropout refers to using Dropout during the forward pass of the network. Random units are dropped out while making a prediction with a trained network. When doing this multiple times it results in multiple different results. The distribution of these samples represents the uncertainty of the model. Monte-Carlo Dropout is shown to approximate a Deep Gaussian Process. The proof requires that the dropout is also used on every layer during training (Gal and Ghahramani 2016).
This method can be applied to a lot of existing models that already use dropout without retraining, and with little modification to any Neural Network, but having to retrain it.

### 2.2.3. Concrete Dropout

The same authors that proposed Dropout as an uncertainty estimate (Monte-Carlo Dropout) (Gal and Ghahramani 2016) also improved upon this concept with Concrete Dropout. The goal is to produce a well-calibrated model using Dropout for uncertainty by training not only the traditional Neural Network parameters, but also the Dropout probabilities. Concrete Dropout will try to fit the estimates to the parameters automatically during training. As a result the estimated uncertainty will be lower with more data, which is not the case with standard Monte-Carlo Dropout (Gal et al. 2017; Melis et al. 2018).
The approaches using Dropout estimate Epistemic Uncertainty by introducing randomness (via Dropout) that collapses for observed values while causing uncertain predictions for data far from the training data.

### 2.2.4. Deep Ensembles

Lakshminarayanan et al. (2017) propose to use ensembles as a way to get an Epistemic Uncertainty estimate, meaning multiple models are trained with the same data. The variance of the ensemble's predictions is used as an Epistemic Uncertainty estimate. The models are initialized randomly and independently and are trained with differently shuffled training data each. The models learn the same function from the same data, but the randomness introduced by the initialization and the shuffling yields different results for unknown data. Intuitively this results in likely higher Epistemic Uncertainties for data that is further away from the training data. For (Heteroscedastic) Aleotoric Uncertainty the authors propose learning it as model outputs similar to MDNs (see Section 4.2). As an extension it is proposed to use adversarial training to improve robustness as well as

smooth the predictive distribution (this is because it ensures that similar examples will have similarly confident predictions).
As opposed to other methods introduced here, there is no mathematical proof (in the paper) that ties this method to Gaussian Processes or the Bayesian Learning framework, but in tests the method performs well and often better than Monte-Carlo Dropout which it has been compared to (Lakshminarayanan et al. 2017). It is also very simple, fast, and easy to parallelize.

### 2.2.5. Bayesian Neural Networks

Bayesian Neural Networks (BNNs) are the result of applying the Bayesian Learning framework to Neural Networks. In Bayesian Learning the model parameters are represented by probability distributions, expressing the likelihood of different values of the parameters. Learning requires a prior distribution over the parameters, usually based in background knowledge. The learning process itself consists of updating the distributions with the observations using Bayes' rule (MacKay 1992). In the application of Bayesian Learning to Neural Networks the parameters are the weights and biases of the network. However, it is very hard to apply any prior beliefs to these parameters, but there are some initial distributions that are shown to work well (Neal 2012).
As the parameters are represented by probability distributions the output of such a network will also be a probability distribution. In practice the network works by sampling the distributions and therefore approximating them. Recent research has made this kind of network computationally feasible (Blundell et al. 2015), but still considerably slower than a classical Neural Network approach.
A Bayesian Neural Network gives an approximation of the Model Uncertainty (Epistemic Uncertainty), similar to the previously introduced Dropout methods. In fact the Dropout methods can be thought of as Bayesian Neural Networks because they impose a distribution over their parameters via dropout (Gal and Ghahramani 2016). For the Aleatoric Uncertainty it is proposed to learn it separately for the Homoscedastic case and to learn parameters for a distribution as model outputs for the Heteroscedastic case (Gal 2016; Kendall and Gal 2017). Similar to what is done with Mixture Density Networks.

### 2.2.6. Deep Gaussian Processes

A Gaussian Process models a distribution over functions. They are used in Machine Learning by using Bayes' rule to update the distribution with new observations like it is done in Bayesian Neural Networks (in fact Bayesian Neural Networks converge to a specific Gaussian Process (Wilson et al. 2016)). The prediction is also a distribution. Learning with a Gaussian Process is a kernel method. It uses a kernel function, which is used to model certain assumptions about the data (e.g. periodicity) (Wilson et al. 2016; Williams and Rasmussen 2006). It has also been shown that given the right conditions a Neural Network with one layer containing infinite units is equivalent to a Gaussian Process (Neal 2012).

The idea of Deep Gaussian Processes (Deep GPs) is to stack Gaussian Processes in a similar manner as it is usually done for Neural Networks. This method was tested for digit recognition where the quality improved with more layers, which indicates that the model is able to learn structures in a similar way to classical Neural Networks (Damianou and Lawrence 2013).

Gaussian Process methods will learn the Epistemic Uncertainty, the Aleatoric Uncertainty is usually learned independently.

### 2.2.7. (Functional) Neural Processes

The authors of Garnelo et al. 2018b developed a method called Neural Processes (NPs). Like Deep Gaussian Processes it aims to combine advantages of Neural Networks and Gaussian Processes. The approach is to model distributions over functions the same way Gaussian Processes do with a Neural Architecture. This way the training is cheaper than with Gaussian Processes and scales better with more data.

The way Neural Processes are designed is that a latent gaussian distribution is used, parameterized by an encoder network. A decoder network uses the distribution with the input data to make a prediction. The latent distribution represents a prior and is the equivalent of a kernel in Gaussian Processes. In contrast to Gaussian Processes, this distribution also needs to be learnt. This means instead of choosing a kernel, ideally based on prior knowledge of the problem, the Neural Process can be pre-trained with similar data. This is good for problems where there is only a small amount of training data available but the model can be pre-trained with other data of the same domain. The model will have learnt a prior fitting the application and give good predictions with little amounts of data. Of course this requires that the pre-training was done with data sufficiently relevant to the problem at hand (Garnelo et al. 2018b; Kim et al. 2019; Garnelo et al. 2018a).

A recent extension to Neural Processes are Functional Neural Processes (FNPs) (Louizos et al. 2019). In contrast to Neural Processes, Functional Neural Processes do not use a single global latent variable, instead they use local latent variables generated by the encoder Network. The model constructs a graph in the embedding space using some

metric (e.g., proximity in the embedding space). The latent variables important to the inputs parameterize the predictive distribution (the importance is determined through the graph). The model is a Bayesian model that learns its priors from the data. The idea behind Functional Neural Processes is similar to Neural Processes: learn a prior distribution as part of the model. This is motivated by Gaussian Processes where kernels can be chosen based on prior knowledge of the domain as well as Bayesian Learning in general where a prior distribution is part of the learning process. This aims to solve the problem of choosing priors for Bayesian Neural Networks where the meaning of the weights and biases is not clear and therefore it is very hard to make any assumptions about the prior distributions. Functional Neural Processes learn prior distributions by inferring a structure (the graph) over the encoding space using some metric and using the relationships the graph represents. The authors state it performs competitively and delivers more robust uncertainty estimates than Neural Processes (Louizos et al. 2019).
(Functional) Neural Processes internally learn latent distributions. The models defined in Louizos et al. (2019) and Garnelo et al. (2018b) the loss function partly consists of the log likelihood of training data to be in the output distribution. The models will therefore capture Aleatoric Uncertainty as well as Epistemic Uncertainty.

## 2.3. Evaluation Metrics

Besides the commonly known metrics used for evaluating predictions we also need some metrics for the quality of the predictive uncertainty. This is more complicated than the normal error metrics because the assumed true distribution, the observed data is drawn from, is not known.

### 2.3.1. Calibration and Sharpness

In Gneiting et al. (2007) it is proposed to evaluate probabilistic forecasts by **calibration** (also called reliability (Bröcker 2012)) and **sharpness**.

**Calibration** Measures the statistical consistency between predictive distribution and observation (Gneiting et al. 2007) (i.e., how well the predictive distributions explain the observations).

**Probabilistic** The probability integral transform (PIT) value is uniform. The PIT value is defined as $p_t = F_t(x_t)$ where $F_t$ is the predictive distribution at $t$ and $x_t$ is the observation at $t$. For an ideal forecaster $p_t$ is uniform. This is usually analyzed with a histogram of the PIT values. The histogram gives an overview of the quantiles of the predictive distributions the observations fall in. If the forecaster captures the real distribution accurately the histogram is uniform.

**Marginal** The consistency between the overall predictive distribution and empirical distribution of the data. Obtained by taking the difference between the average predictive cumulative density function (CDF) and the CDF of the empirical observation distribution. An ideal forecaster predicts the real distribution of the data accurately, and the difference between the CDFs converges to zero (with a growing amount of data).

**Sharpness** Producing forecasts as sharp as possible while also being calibrated. Measured by the average width of prediction intervals. The most confident forecaster, while also accurately representing the distribution of the data, is considered the best.

These metrics in combination measure the quality of a probabilistic forecaster, in that they would be ideal for an ideal forecaster, but they do not guarantee an ideal forecaster.

### 2.3.2. Proper Scoring

Proper scoring rules give a numerical score for the quality of the probability estimates and encourage calibrated and sharp predictions (Gneiting et al. 2007). The following definitions are for scores as penalties that reach a minimum for the ideal case, as used in (Gneiting et al. 2007). A proper scoring rule $S(F, x)$ with predictive distribution $F$ and observation $x$ is defined as always having a smaller or equal expected score to the score of any other predictive distribution, if the predictive distribution is the real distribution. So for $S(F2, F1)$ being the expected score of $F2$ assuming the outcome distribution is $F1$ then $S$ is proper if $S(F1, F1) \leq S(F1, F2)$ with $F1 \neq F2$ (Tsyplakov 2013; Gneiting et al. 2007). As a proper scoring rule we use the Continuos Ranked Probability Score (CRPS) shown in Equation (2.1). It is robust and the generalization of the absolute error (Gneiting et al. 2007).

$$crps(F, x) = \int_{-\inf}^{\inf} (F(y) - 1(y \geq x)^2) dy \tag{2.1}$$

# 3. Related Work

In this chapter we explore some research in the field of probabilistic forecasting using Deep Learning.

In Wang et al. 2017 a group of researchers have employed Deep Learning in a probabilistic manner in wind power forecasting. They use wavelet decomposition on the data and arrange it in a two dimensional array. This allows them to adopt Convolutional Neural Networks (CNNs) as part of their model. CNNs have shown great results in the area of image recognition and feature extraction of two dimensional data. The outputs of multiple individual well-trained CNNs is used to estimate Model Uncertainty (Epistemic Uncertainty) by assuming a gaussian distribution and calculating mean and variance from the outputs (Ensembling). These CNNs are trained with different hyper-parameters and inputs. The inputs for the different CNNs are varied by using a different number of previous values. An estimate for the Data Uncertainty (Aleatoric Uncertainty) is calculated as the average noise of the difference between prediction and input data (Wang et al. 2017).

Dahua et al. 2018 employ a Neural Network based model to do load forecasting in a probabilistic fashion. To achieve this a Quantile Regression approach is used. The input data is made up from the linear trend of the load, the temperature, and a number of one-hot encoded features like time of day, day of week, and holiday. These one-hot encoded features are embedded by multiplying them with an embedding matrix and are then flattened. This approach reduces the number of inputs and allows weighing the features differently. Using Quantile Regression for the Neural Network an uncertainty estimate is given for the load forecast by producing a number of quantiles for the output. This approach also takes into account input uncertainty for the temperature. This is done by taking a number of estimates for the temperature, based on values from previous years, and running the forecast for each of them. The predictive distribution is calculated from the outputs of all the passes. The authors compare their method to Multiple Linear Regression, a simple Neural Network, and a Linear Quantile Regression model. They conclude that their method is better than the ones compared to (Dahua et al. 2018). We previously discussed that Quantile Regression will only give an estimate of the Aleatoric Uncertainty, that is the uncertainty that is inherent in the data, so this work ignores the Epistemic Uncertainty.

With Zhu and Laptev (2017) Engineers at Uber developed an approach to time series prediction that uses Monte-Carlo Dropout. They find that for their application of extreme event forecasting the generated uncertainty estimate of using Dropout is not good enough. They argue that outliers, particularly ones that are not at all captured in training data, should result in larger uncertainties than the given Model Uncertainty. Their way of solving this is using an Autoencoder to embed the input features, the embedded features are then fed into the network making the prediction. The uncertainty estimate is given through applying Dropout to both the Autoencoder and the prediction network. The authors call this additional uncertainty Model Misspecification. This uncertainty will be high if there are anomalies due to the embedded features being outliers in the embedding space. It is stated that the predictions work well with their data and are particularly good for anomalous events (like holidays) (Zhu and Laptev 2017).

Researchers at Amazon have also taken on the problem of probabilistic forecasting using Deep Learning (Salinas et al. 2017). Their approach consists of a Long Short-Term Memory Neural Network that produces a forecast based on a number of previous points of a time-series. The network outputs a distribution of the next point of the time-series. This is a similar approach to Mixture Density Networks and will only capture the Aleatoric Uncertainty. To produce multi-step ahead forecasts the authors feed the model with the output of the previous forecast(s) as well as data from previous time steps (if available). As the outputs are distributions, they are sampled to incorporate the uncertainty into the next forecast. The authors compare their method to other (non Deep Learning) methods and "*find improvements of around 15%*" (Salinas et al. 2017).

It is apparent that, at the time of writing, the application of probabilistic extensions for Deep Learning in a forecasting context is only sparsely researched. The research in the context of energy time series forecasting is even more sparse. Furthermore, most of the work in this field does not focus on a combined uncertainty estimate of Aleatoric and Epistemic Uncertainty, with the exception of Wang et al. (2017), although only very simple estimates are used. The approaches taking Epistemic Uncertainty into account focus on simple methods like Dropout and Ensembling (Wang et al. 2017; Zhu and Laptev 2017). Zhu and Laptev (2017) in particular focus on an improved estimate for Epistemic Uncertainty (calling it Model Misspecification) for extreme events, by introducing more randomness through an Autoencoder with Dropout. None of the discussed research use more advanced methods like Concrete Dropout, Bayesian Neural Networks, or even Deep Ensembles as described in Lakshminarayanan et al. (2017).

# 4. Methodology

After exploring existing methods, and their usage in forecasting, this chapter explores the methodology of the application to energy time series forecasting.

## 4.1. Method Selection

Not all introduced methods are used for evaluation purposes. We will exclude Mixture Density Networks and Quantile Regression for Neural Networks, because they only give an estimate of the Aleatoric Uncertainty. Both methods can be applied to the other models an therefore will not be applied on their own (more in Section 4.2).

The methods selected for further use are Concrete Dropout, Deep Ensembles, Bayesian Neural Networks, Deep Gaussian Processes, and Functional Neural Processes.

Monte-Carlo Dropout is a simple and mathematically proven method and is a popular technique in related research. Concrete Dropout is chosen as an advanced development based on Monte-Carlo Dropout. As a similarly simple approach we choose Deep Ensembles. We opted against implementing the adversarial training extension from the paper, as it could potentially be used to improve any of our methods. Furthermore, we opted for an implementation of Bayesian Neural Networks applying the concept of distributions over weights and biases directly by sampling from those distributions during forward passes and getting concrete weights and biases, changing the parameters of the distributions via back-propagation. Deep Gaussian Processes are selected as a deep model on the basis on, for probability estimation very popular, Gaussian Processes. Functional Neural Processes are a method that, via a Neural Architecture, aims to model what Gaussian Processes do well. In our case we also prefer the Functional Neural Processes over other types of Neural Processes as it is aimed to work with a single data-set, while Conditional Neural Processes and (Attentive) Neural Processes really excel with a large amount of different data-sets for pre-training and a small data-set for training.

## 4.2. Combined Uncertainty Estimate

We previously excluded the methods that only give an estimate of the Aleatoric Uncertainty and chose methods giving Epistemic Uncertainty estimates, this does not mean Aleatoric Uncertainty should be ignored. It is also part of the general confidence of a prediction. If there is uncertainty in the training data it should be assumed that the situation predicted is uncertain to the same degree. To estimate the Aleatoric Uncertainty we model it as an output of the model in question, as proposed for the Heteroscedastic case in Kendall and Gal (2017) and Lakshminarayanan et al. (2017). As for forecasting it is expected to have different distributions of the training data at different points in time.

The concept is the same as the one used for Mixture Density Networks but we will assume the distribution of the data to be representable as a single gaussian. To make our models learn the uncertainty in the data we make them predict two values: the mean of and the standard derivation of a gaussian distribution. To train the models we maximize the value of the probability density function at the training data points (i.e., the probability of the training data to be in the output distribution). Because the calculation is numerically more stable and losses are minimized we define the loss as follows:

$$\mathcal{L} = -\log\left(pdf(x)\right) = -\frac{(x-\mu)^2}{2\sigma^2} - \log\sigma - \log\sqrt{2\pi} \tag{4.1}$$

Equation (4.1) is a simplified version of the loss function used for Mixture Density Networks in Bishop (1994) with only one gaussian instead of a mixture and has been proposed to estimate mean and variance using Neural Networks in Nix and Weigend (1994). The constant term can be ignored.

For most methods that give an estimate of the Epistemic Uncertainty a number of samples are drawn, and the parameters of the distribution of the Epistemic Uncertainty are inferred from the distribution of the samples. When learning Aleatoric Uncertainty as model outputs, this will result in a mixture of gaussian distributions. Kendall and Gal (2017) and Lakshminarayanan et al. (2017) propose a way to combine the two uncertainty types to form the predictive distribution as a single gaussian with predictive mean and variance:

$$\mu_p^2 \approx \frac{1}{T}\sum^{T}\mu_t \tag{4.2}$$

$$\sigma_p^2 \approx \frac{1}{T}\sum^{T}\mu_t^2 - \frac{1}{T}(\sum^{T}\mu_t)^2 + \frac{1}{T}\sum^{T}\sigma_t^2 \tag{4.3}$$

Equation (4.2) and Equation (4.3) give approximations for the predictive mean $\mu_p$ and the predictive variance $\sigma_p^2$ where $T$ is the number of samples and $\mu_t$ and $\sigma_t$ are the parameters of the Aleatoric distributions the model predicted for each sample.

For the most part this is how we get a combined uncertainty estimate for our used methods. With the exception of Functional Neural Processes, in the version proposed and implemented in Louizos et al. (2019), they already learn latent distributions that are trained with similar terms as Equation (4.1) and therefore will combine Aleatoric and Epistemic Uncertainty on their own.

## 4.3. Load Forecasting

To apply our methods to the domain of energy time series we consider the problem of load forecasting, specifically short-term (one-step) forecasting and day-ahead forecasting.

As reference models outside of the domain of probabilistic Deep Learning, we chose a simple (Linear) Quantile Regression model as well as a Neural Network trained to estimate the Aleatoric Uncertainty by predicting mean and variance as described in Section 4.2. For the Quantile Regression model, the predictive uncertainty is a distribution estimated from the predicted quantiles.

### 4.3.1. Parameters

To optimize the models' hyper-parameters, Bayesian Optimization along with three-fold cross validation is used. To get good parameters for probabilistic forecasting the configurations are scored with CRPS (Equation (2.1)).

We optimize most of the methods' parameters including layer sizes, network depths, learning rates, number of training epochs, and other model specific parameters. The optimization is done separately for the two forecasting scenarios that we examine. To keep the optimization feasible, we optimize some of the parameters independently of each other (like learning rate and layer sizes). For other parameters we use best practices and values that have worked well in our experience. In particular Deep Ensembles, Concrete Dropout, and Bayesian Neural Networks are trained with the same layer configuration as the reference Neural Network model, for which the parameters are optimized. Furthermore, Deep Gaussian Processes prove to scale badly and take a long time to converge for big amounts of data, so the parameters could only be optimized to some degree regarding optimization time. Similarly, for Functional Neural Processes there is a large amount of additional hyper-parameters that prove sensitive in terms of convergence speed and accuracy. The parameters for Deep Gaussian Processes and Functional Neural Processes could not be optimized to the same degree as for other methods. A more in depth explanation of the optimization and parameters used can be found in Appendix A.3.

### 4.3.2. Pre-Processing

As a general pre-processing step we standardize and de-seasonalize the data for days and weeks, as proposed in (Zhang and Qi 2005) for Neural Network forecasting. We find that, with the features constructed for the forecasting scenario (Section 4.3.3), the de-seasonalization only slightly improve convergence times.

### 4.3.3. Feature Construction

For forecasting, we, empirically and through best practices, constructed a number of features all the models use. First of all, the time of day and time of year, as proposed in Hong et al. (2016), tranformed with a sinus function to model that, for example, times late at night and early in the morning are similar. Additionally, lagged variables for minutes, hours, and days before the point in time the forecast is being made for (Hong et al. 2016) are introduced. To capture the more anomalous loads, we one-hot encode special days (i.e., holidays) and special day adjacent days / bridging days, motivated by the research in Arora and Taylor (2018). The day of the week is also one-hot encoded in the features which improved predictions. A complete list of features can be found in Appendix A.2.

### 4.3.4. Forecasting Scenarios

Generally we consider point forecasting of electricity loads with two variations:

- With short-term lagged variables for the immediate future. Effectively forecasting the load for 15min in the future.
- Without short-term lagged variables, meaning the resulting model can be used to forecast the next 24h without feeding the predictions back into the model (day-ahead). Forecasting the 96 values of the next 24h means running the forecast 96 times.

We find that it was not feasible to model the predictions for the next 24h via model outputs, as that would mean nearly 200 outputs (with mean and variance for each time step at a resolution of 15 minutes). This leads to slower convergence and in general requires more data. For Deep Gaussian Processes in particular, nearly 200 Gaussian Processes are needed in the last layer, at that point the Deep Gaussian Processes tend to not converge at all and become generally very slow to train.

We also experimented with feeding the predictions back into the model as the short-term lagged variables. This works to some degree but, for the full 24h, performs the same or worse than the version without the short-term lagged variables as the error aggregates over the predictions.

### 4.3.5. Post-Processing

In Section 2.3 we introduced metrics of calibration and sharpness and explained that, to evaluate based on sharpnesss, the forecaster has to be well calibrated. Calibration refers to how accurately the forecaster represents the distribution in the data. Common characteristics of badly calibrated probabilistic forecasts are overdispersion, underdispersion, and bias. Overdispersion refers to the forecaster generally estimating too much variability in the predictions, while underdispersion means the forecaster estimates too little variability. A bias hints to a forecaster, on average, over- or underestimating (Thorarinsdottir et al. 2016; Hamill 2001).

To correct for badly calibrated models we introduce a simple post-processing recalibration step that fits two parameters, one that adds to the mean of the predictive distribution and one that scales the variance. These parameters are optimized independently for each model using the training data minimizing the CRPS, similar to what is proposed in Gneiting et al. (2005) and Thorarinsdottir et al. (2016).

### 4.3.6. Evaluating Epistemic Uncertainty

The metrics introduced in Section 2.3 are well established but do not give insight into the quality of the Epistemic Uncertainty estimates (the main difference between most of the methods is how they produce an Epistemic Uncertainty estimate). The metrics only evaluate in what way the models have certain properties of an ideal forecaster, which knows the true distribution of the data. With Epistemic Uncertainty we want to quantify what the model does not know, meaning, how well the model recognizes when new data is far from what was learned before.

For evaluating Epistemic Uncertainty we opt for a similar solution as Lakshminarayanan et al. (2017), trying to make predictions on sufficiently different data, not from the original data-set and visualizing the distribution of the predictive standard deviations. To make sure the data is different enough, we synthetically generate out of distribution (o.o.d.) data by using the test data and generating data that randomly deviates from it. The deviation is consistently kept above a minimum value to ensure the generation of data that is sufficiently different from normal load data. We pre-process and generate features for the synthetic data the same way as for the original load data, with consecutive timestamps starting at a random point in time. The pre-processing has the effect that the indicator variables constructed (like one-hot encoded special days etc.) are not anomalous, which would still be the case even if an anomalous situation occurs.

# 5. Evaluation

In this chapter we evaluate the models for a short-term forecasting scenario and a day-ahead forecasting scenario. For the day-ahead case we opted to train the models the same way as for the short-term scenario but without the short-term lagged input variables (see Section 4.3.4). All computations are done using a NVIDIA Tesla V100 GPU with 32GB of graphics memory. The software is written in Python using PyTorch (Paszke et al. 2019) as a Deep Learning framework (for more information on the implementation refer to Appendix A.1).

## 5.1. Data

To train and test our models we use energy load data for Germany from Open Power Systems Data (*Open Power System Data. 2019. Data Package Time series. Version 2019-06-05* 2019), more specifically the load data from the ENTSO-E Power Transparency (Figure 5.1) with a resolution of 15 minutes.

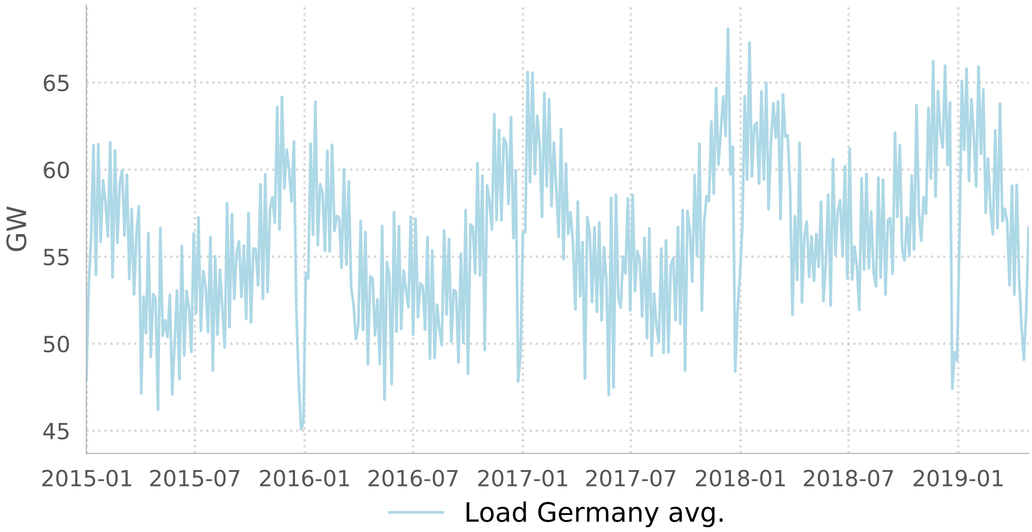


**Figure 5.1.** Data used for load forecasting scenarios. Load data for Germany from beginning of 2015 to early 2019. Re-sampled to 4 day intervals for clarity.

The data includes around 140,000 load values, a fifth of it (a little less than one consecutive year) is used as a test set, the remaining data is used to choose the hyper-parameters via cross validation and train the final models.

## 5.2. Load Forecasting Results

All evaluations are done on the test set of around 30,000 data points, which were not used in training. We use the metrics introduced in Section 2.3 and compare the methods in terms of calibration and sharpness, later we also assess the Epistemic Uncertainty capabilities of the methods in terms of anomalous data. Even though we use scaling and de-seasonalization in pre-processing all values and plots are produced after an inverse transformation of the predicted values back to their original scale (GW).

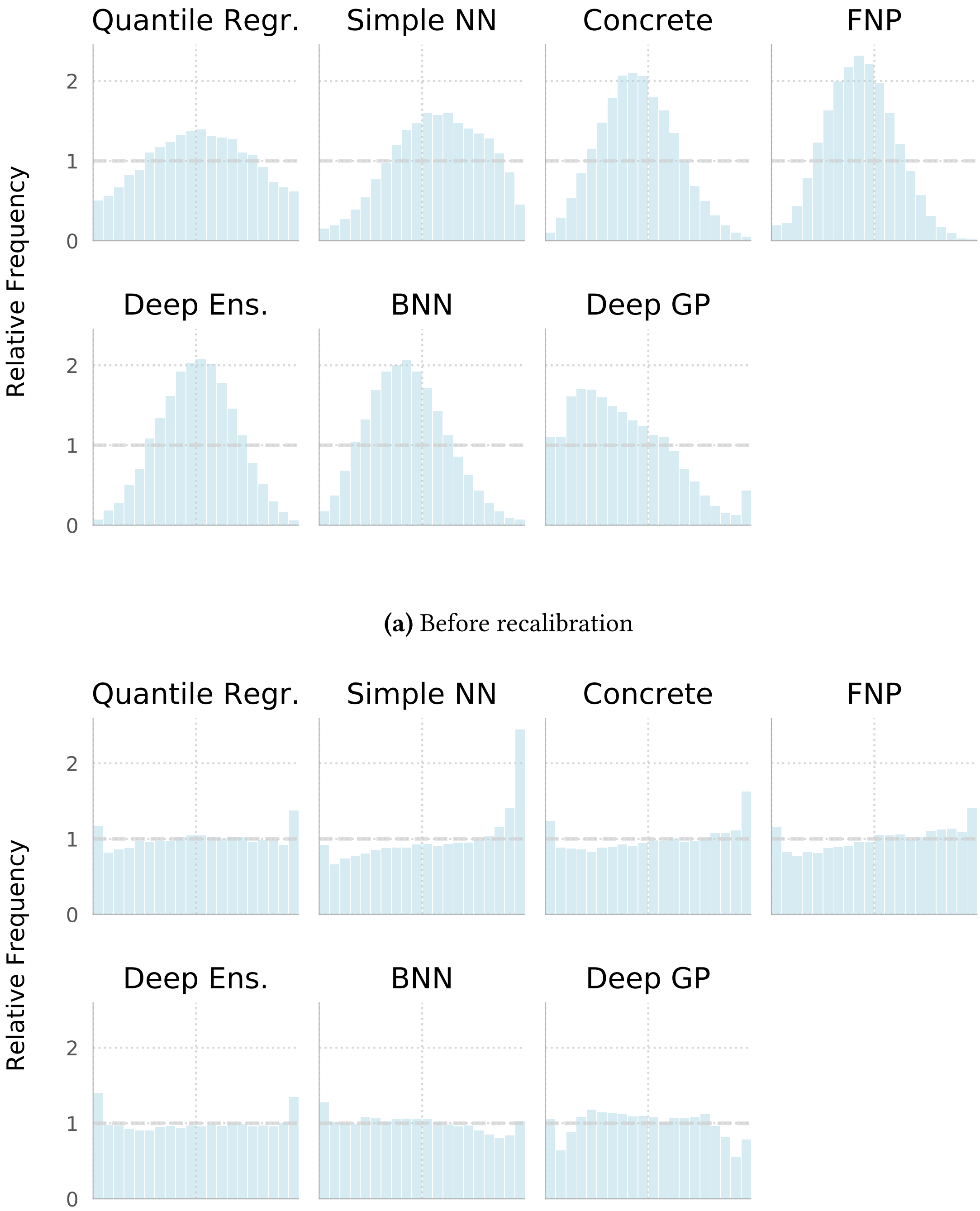
### 5.2.1. Short-Term Forecasting

We evaluate the short-term forecasting scenario first. For this scenario the models were trained to predict the next value of the electricity load time series. In Figure 5.2a the probability integral transform (PIT) histograms are shown, for a well probabilistically calibrated model the PIT histogram should be uniform.

We observe that the reference Quantile Regression model is the closest to a uniform distribution out of the box. All models show signs of overdispersion (the PIT historams are bell-shaped), meaning there is too much variability in the predictions. Intuitively the PIT histogram shows the quantiles of the predictive distributions in which the observed real data tends to fall. A bell-shape suggests that the observations tend to be in the central quantiles of the predictive distributions. For an ideal forecaster the observations should overall match the predictive distributions and be evenly distributed over the quantiles resulting in a uniform PIT histogram.

For the Simple Neural Network, the Functional Neural Process, and in particular the Deep Gaussian Process there is a skew in the histograms. A skew is a sign of a biased forecaster, meaning the forecaster tends to over- or underestimate.

In Figure 5.2b the PIT histograms after recalibration are shown. They now exhibit properties much closer to a uniform PIT. In terms of probabilistic calibration they are mostly well calibrated with the simple reference Neural Network having a slight, more complicated bias that cannot be removed by shifting the mean of the model. The following results all consider the recalibrated models.



**(a)** Before recalibration

**(b)** After recalibration

**Figure 5.2.** Probability integral transform (PIT) histograms for the tested methods in the short-term load forecasting scenario. Showing the probabilistic calibration before and after recalibration. The grey line in each of the plots represents an ideal uniform PIT histogram for a well calibrated forecaster.

Figure 5.3 shows plots to evaluate marginal calibration (i.e., how consistent the overall predictive distribution is with the empirical distribution of the test data). To achieve this, we plot the difference between the average predictive CDF and the empirical CDF of the test data. For marginal calibration the differences should be as small as possible, that is, the plot should be as close to zero as possible. Considering marginal calibration the best methods are the reference Quantile Regression, the Concrete Dropout model and the Deep Ensembles, with the Bayesian Neural Network model following closely. The simple reference Neural Network, the Functional Neural Process and the Deep Gaussian Process models perform equally bad in terms of marginal calibration.

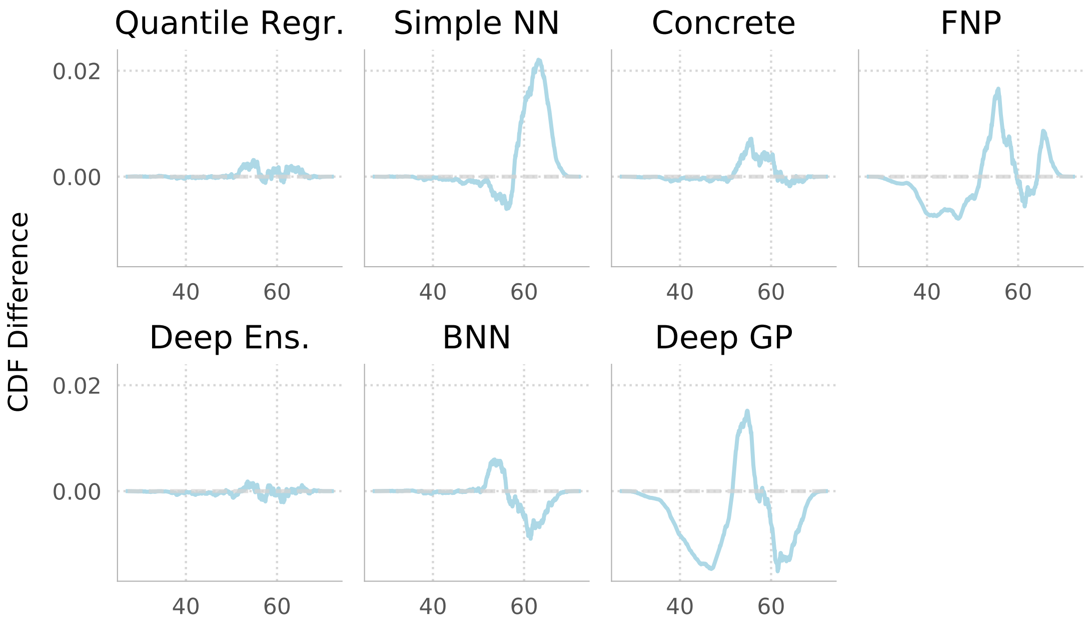


**Figure 5.3.** Plots to evaluate the marginal calibration of the methods for the short-term load forecasting scenario after recalibration. The plot shows the difference between the average predictive CDF and the CDF of the empirical distribution of the observations.

The box-plots in Figure 5.4 show the sharpness of the predictions (i.e., the general confidence of the models). The plots show the distributions' 90% confidence interval widths of the methods. The box plot depiction is chosen as proposed in Gneiting et al. (2007) for a heteroscedastic forecast. The optimal case is for the model to be as confident as possible, while also accurately representing the real distribution of the data (sharpness under calibration (Gneiting et al. 2007)).

The sharpest models are Deep Ensembles and Concrete Dropout, with Deep Ensembles being generally sharper but with the distribution being skewed. We observe similar

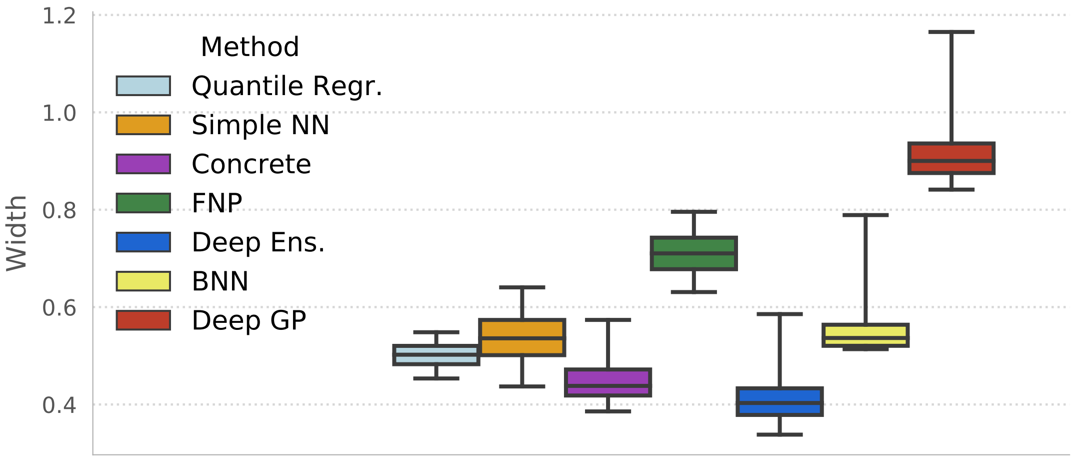


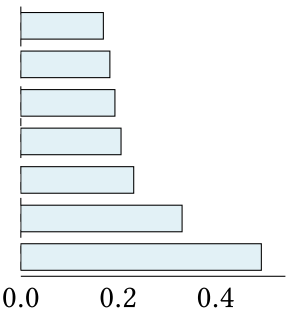
**Figure 5.4.** Distribution of widths of the 90% confidence intervals of the model predictions for the short-term load forecasting scenario after recalibration.

behavior for the Bayesian Neural Network and the Deep Gaussian Process models. The Bayesian Neural Network model is similarly sharp as the reference models but, in terms of sharpness, is considered worse because the upper limit for the confidence interval widths is higher.

We list the root mean squared error (RMSE), the mean absolute percentage error (MAPE), and the continuous ranked probability score (CRPS) in Table 5.1. For all values applies, the lower the value the better the method.

**Table 5.1.** RMSE, MAPE, and CRPS for the short-term load forecasting scenario. The bar chart visualizes the CRPS of the methods for an easier comparison. The table is sorted by CRPS in ascending order.

| | RMSE | MAPE [%] | CRPS |
|---|---|---|---|
| Deep Ensembles | 0.398 | 0.409 | 0.170 |
| Concrete | 0.413 | 0.443 | 0.183 |
| Quantile Regression | 0.464 | 0.462 | 0.193 |
| BNN | 0.471 | 0.494 | 0.206 |
| Simple NN | 0.496 | 0.549 | 0.232 |
| FNP | 1.083 | 0.839 | 0.331 |
| Deep GP | 1.654 | 1.276 | 0.494 |



The metrics confirm our previous observations. Deep Ensembles perform the best, followed by Concrete Dropout and Quantile Regression.

Table 5.2 gives a comparison of the time it takes for each method to be trained and the time the methods take to make predictions for the test data (around 30,000 data points). The time to train for the main models and the simple reference Neural Network is between 2h and 5h, with relatively short prediction times except for the Functional Neural Process model.

**Table 5.2.** Time to train and predict for the models of the short-term load forecasting scenario. The table is sorted by train time in ascending order.

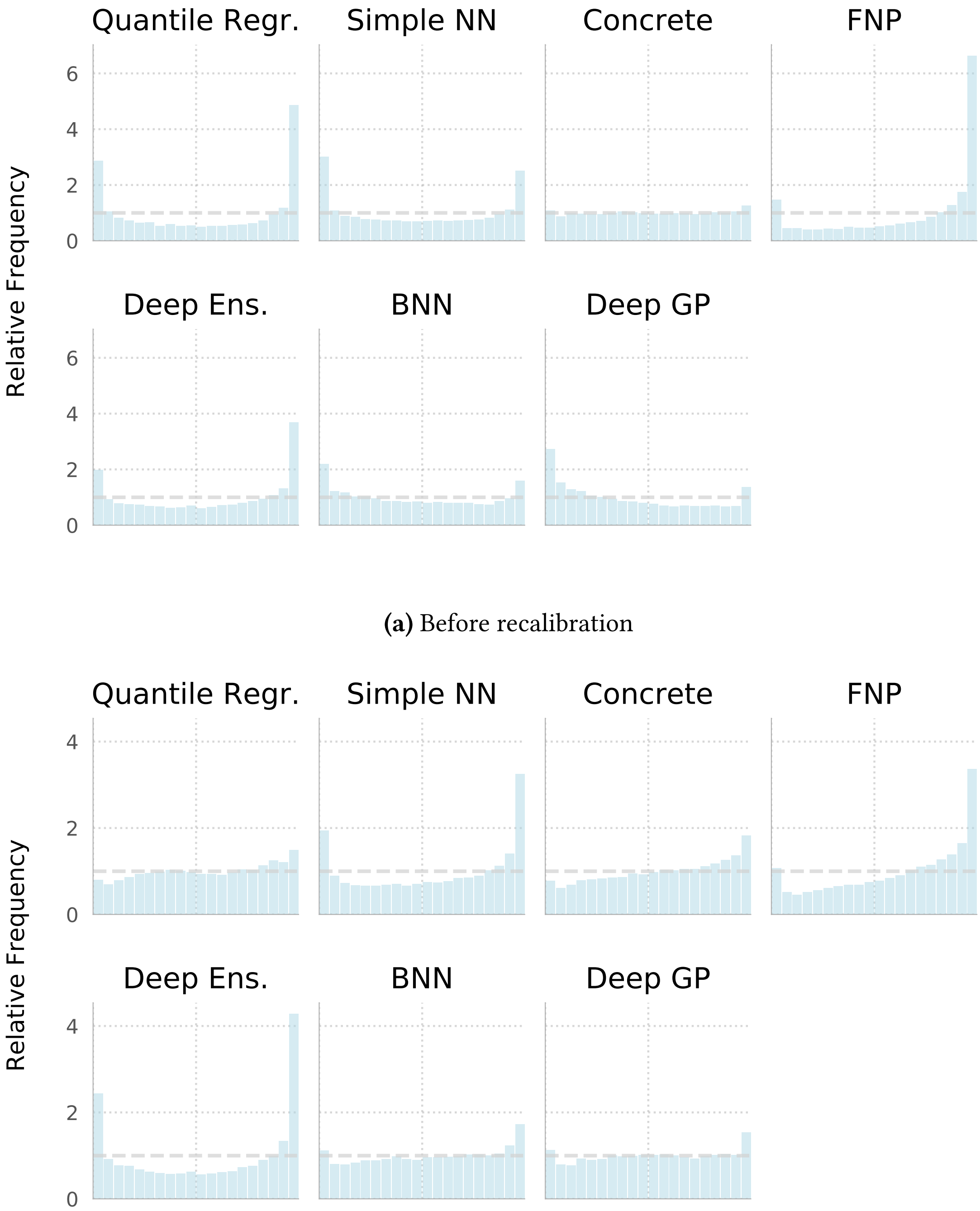
| | Train Time [min] | Predict Time [s] |
|---|---|---|
| Quantile Regression | 0.366 | $6.542 \cdot 10^{-2}$ |
| FNP | 130.103 | 161.449 |
| Deep GP | 171.437 | 0.881 |
| Deep Ensembles | 198.978 | 1.748 |
| Simple NN | 213.891 | 0.762 |
| BNN | 269.421 | 9.820 |
| Concrete | 287.459 | 13.580 |

### 5.2.2. Day-Ahead Forecasting

The day-ahead forecasting is produced the same way as the short-term forecasting but without the short-term lagged variables. This approach allows for a 24h-ahead forecast by running the forecast for each of the points in time for the 24h.

The PIT histograms before recalibration are shown in Figure 5.5a, the histograms after recalibration are shown in Figure 5.5b. The models have been recalibrated the same way as before, all following results refer to the recalibrated models.

In general, the probabilistic calibration results are worse than in the short-term scenario even after recalibration. Only Quantile Regression, the Bayesian Neural Network, and the Deep Gaussian Process are calibrated well with a slight bias to the right. Generally, all the methods have this bias to some degree. All the other models show a U-shape in the histograms, meaning they are somewhat underdispersed (i.e., the models are generally too confident in their predictions). Comparing the PIT histograms before and after recalibration (Figure 5.5a and Figure 5.5b) shows that the recalibration was considerably less effective compared to the short-term scenario, but the models were initially already comparatively well calibrated.



**(a)** Before recalibration

**(b)** After recalibration

**Figure 5.5.** Probability integral transform (PIT) histograms for the tested methods in the day-ahead load forecasting scenario. Showing the probabilistic calibration before and after recalibration. The grey line in each of the plots represents an ideal uniform PIT histogram for a well calibrated forecaster.

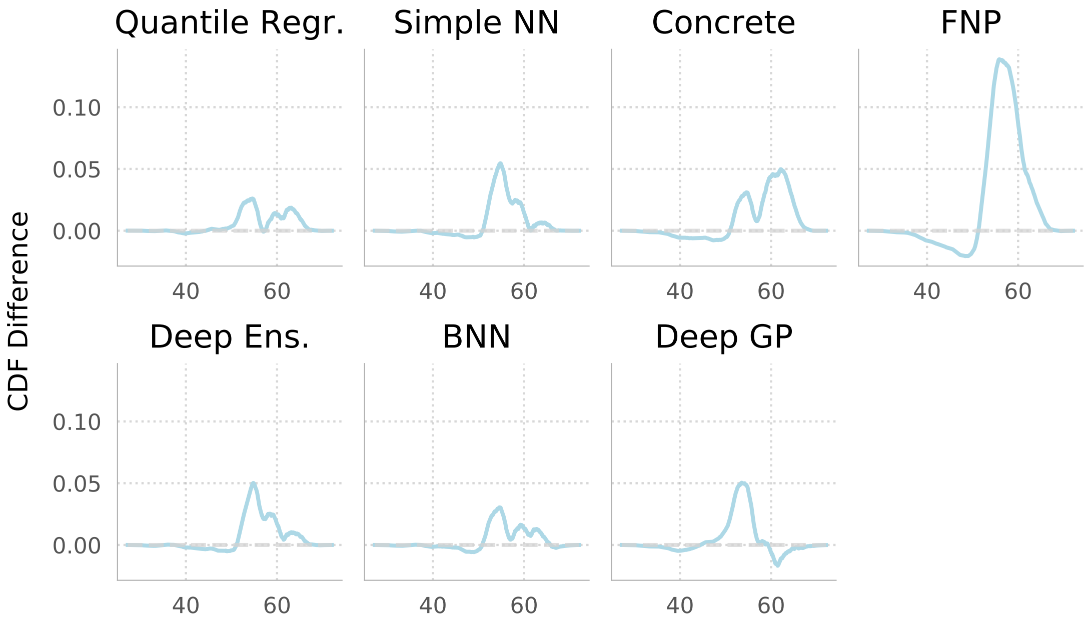


**Figure 5.6.** Plots to evaluate the marginal calibration of the methods for the day-ahead load forecasting scenario after recalibration. The plot shows the difference between the average predictive CDF and the CDF of the empirical distribution of the observations.

The marginal calibration plots (Figure 5.6) look very similar for Quantile Regression, the simple Neural Network, Concrete Dropout, the Deep Ensemble, the Bayesian Neural Network, and the Deep Gaussian Process model. The Functional Neural Process model performs considerably worse than the others for this metric.

In terms of accuracy metrics the Bayesian Neural Network model performs the best for day-ahead forecasting, closely followed by Concrete Dropout, the reference models, and Deep Ensembles. The Deep Gaussian Process model also performs relatively well in this scenario, as opposed to the short-term case (see Table 5.3). It is of note that, while for the short-term case all metrics ranked the methods in the same order, in the day-ahead case the Deep Ensemble would be about on-par with the simple reference Neural Network if ranked by MAPE or RMSE, beating Quantile Regression. The Deep Ensemble model also is the sharpest (Figure 5.7) in this case closely followed by the simple reference Neural Network, the Bayesian Neural Network, and the Concrete Dropout model, but it is worse in terms of probabilistic calibration (Figure 5.5b).

The Deep Gaussian Process model, while performing relatively well, produces results less sharp than Concrete Dropout, the Deep Ensemble, the Bayesian Neural Network, and

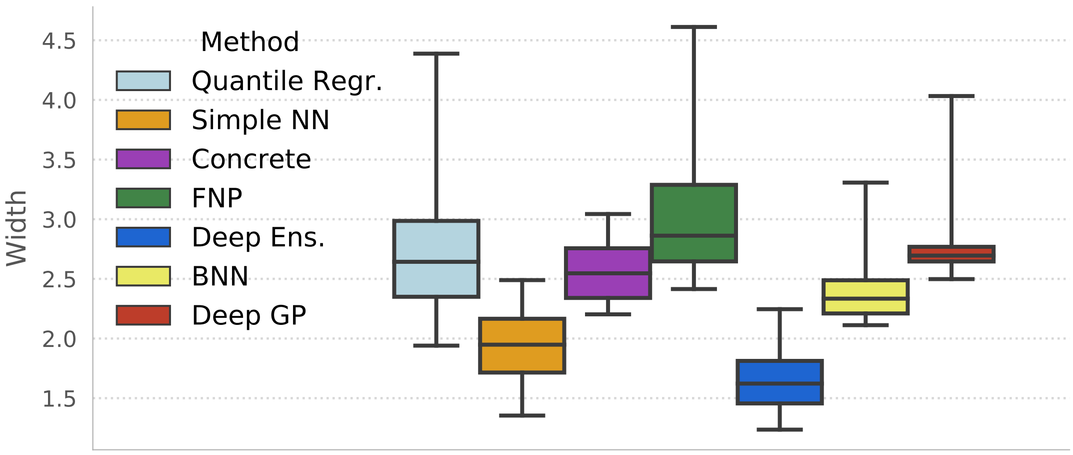


**Figure 5.7.** Distribution of widths of the 90% confidence intervals of the model predictions for the day-ahead load forecasting scenario after recalibration.

the simple Neural Network (Figure 5.7). It is comparable to Quantile Regression and the Functional Neural Process.

The computation times (Table 5.4) are similar to the short-term forecasting case, with the distinction that the simple Neural Network model, the Deep Ensemble, and in particular the Concrete Dropout model take considerably less time to train.

**Table 5.3.** RMSE, MAPE, and CRPS for the day-ahead load forecasting scenario. The bar chart visualizes the CRPS of the methods for an easier comparison. Sorted by CRPS in ascending order.

| | RMSE | MAPE [%] | CRPS |
|---|---|---|---|
| BNN | 1.838 | 2.362 | 0.969 |
| Concrete | 1.905 | 2.475 | 1.010 |
| Simple NN | 1.813 | 2.427 | 1.026 |
| Quantile Regression | 2.108 | 2.619 | 1.054 |
| Deep Ensembles | 1.866 | 2.478 | 1.063 |
| Deep GP | 2.536 | 2.958 | 1.203 |
| FNP | 3.147 | 3.872 | 1.624 |

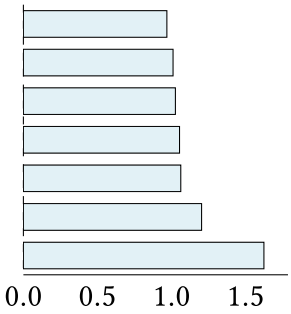


**Table 5.4.** Time to train and predict for the models of the day-ahead load forecasting scenario. The table is sorted by train time in ascending order.

| | Train Time [min] | Predict Time [s] |
|---|---|---|
| Quantile Regression | 0.284 | $6.170 \cdot 10^{-2}$ |
| Concrete | 18.910 | 12.624 |
| Deep Ensembles | 59.091 | 1.863 |
| Simple NN | 64.535 | 0.478 |
| FNP | 131.523 | 160.869 |
| Deep GP | 173.722 | 0.375 |
| BNN | 257.258 | 10.208 |

### 5.2.3. Epistemic Uncertainty Evaluation

To evaluate the Epistemic Uncertainty we use the models from the short-term load forecasting scenario. The models are fed with unseen data and we visualize the standard deviations of the predictive distributions in Figure 5.8. The distributions of the standard deviations over the out of distribution (o.o.d.) data (orange) as well as the test data (blue) are plotted. The further the two distributions are from each other, the better the Epistemic Uncertainty estimate.

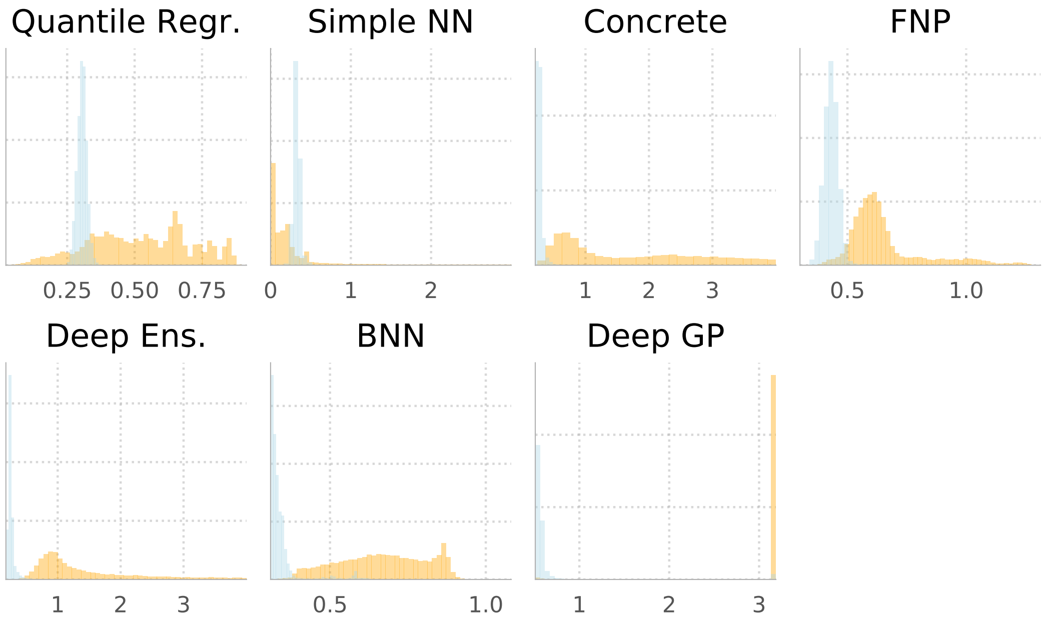


**Figure 5.8.** Comparison of the distributions of the standard deviations for the methods using out of distribution (o.o.d.) load data with non-random indicator variables. Standard deviations of the test set are visualized in blue, standard deviations of the o.o.d. data in orange. Note that, for clarity, the plots for the simple Neural Network, Concrete Dropout, and the Deep Ensemble model have been cut off at 3, 4, and 4 respectively.

For Quantile Regression the confidence for the o.o.d. data is relatively close to the confidence for the test data, it even produces a higher confidence for a part of the o.o.d. data. Similarly, the simple Neural Network produces confidence estimates for the o.o.d. data mostly higher than the confidence of the confidence of the test data, with the confidences trailing off to the right of the plot. Both methods fail to capture the Epistemic Uncertainty well.

The Concrete Dropout model, the Functional Neural Process, the Deep Ensemble model, the Bayesian Neural Network, as well as the Deep Gaussian Process model consistently provide confidences lower and separate to the confidence in the test data predictions. The plots indicate that these methods recognize o.o.d. data well. Deep Gaussian Process in particular are very confident in the recognition of the o.o.d. data.

# 6. Discussion

In the following we discuss evaluation results, complications, and possible reasons for the observed results.

## 6.1. Hyper-Parameter Optimization

To compare the different methods as best as possible we optimize over the hyper-parameters but run into some issues.
We found that, for the data we used, the cross validation scores for each of the validation passes were varying a lot. While the number of data points is not small they still only represent four years of load data, presumably the validation set is not as representative as we would like, and causes the fluctuation. With the scores being less representative of the quality of the configurations tested the Bayesian Optimization will perform worse.
For Functional Neural Processes we found the optimization to be very time consuming because the method has a large number of parameters and some of them cause the model to converge only very slowly and use up a lot of memory. The main cause of this is that Functional Neural Processes internally use a graph of latent variables constructed from a reference set of data. Consequently the graph grows with the reference set size and becomes more expensive in both compute time and memory. This means that for large amounts of data the reference set has to be relatively small, as the mini-batching approach introduced in Louizos et al. (2019) does not batch the graph data. Deep Gaussian Processes scale badly and prove similarly time consuming for parameter optimization.
These limitations result in likely sub-optimal parameters used for the Deep Gaussian Process and Functional Neural Process models.

## 6.2. Calibration and Data Quantity

We observe that for short-term forecasting the probabilistic calibration was overdispersed (Figure 5.2a), which could be easily corrected by the simple recalibration. The data used for the short-term forecasting scenario does not have a lot of inherent uncertainty, meaning the probabilistic component is less important, and well calibrated and sharp forecasts take less effort. The situation is different for the day-ahead scenario. The data is overall noisier with the lack of information. Consequently the Aleatoric Uncertainty is generally higher and more complicated than for the short-term case. The underdispersion of the PIT histograms (see Figure 5.5b) indicates that the models, to some degree, fail to capture the real distribution of the data. That and the fact that this effect remains after recalibration points to the data not being representative enough to better capture or test the Aleatoric Uncertainty (the recalibration is also done using the training data). While Quantile Regression and the Deep Gaussian Process seem well calibrated, they lack in sharpness and accuracy (higher MAPE and RSME) compared to Concrete Dropout, the Deep Ensemble, and the simple Neural Network (see Figure 5.7 and Table 5.3). We conclude that to improve the calibration of the forecasters more training data is needed.

## 6.3. Comparison of Approaches

The following explores the evaluation results and discusses underlying effects in the context of comparing the different approaches.

### 6.3.1. General Performance

We found in the evaluation (Chapter 5) that the simple reference Quantile Regression model performs well on the problems at hand. The fact that the Quantile Regression models performs well in both scenarios indicates that the problem is relatively simple. It is always advisable to make a problem as simple as possible for Machine Learning no matter how powerful a method is. If the data is pre-processed well and the features are constructed intelligently the method has to learn less. In our scenarios the feature construction, de-seasonalization and standardization of the data takes a lot of complexity out of the problem. At the same time, Deep Learning methods excel at highly non-linear function approximation and finding complex relationships in data, if there are less complex relationships the Deep Models loose some of their edge.

That said, the simpler neural models perform better or similarly to Quantile Regression. Concrete Dropout, Deep Ensembles, and Bayesian Neural Networks perform particularly well. Intuitively they should perform very similarly to the simple reference Neural Network, with the only difference being the Dropout, the ensembling, or the distribution over weights ans biases respectively. The structure, activation functions, and learning process are the same. By introducing some randomness they tend to avoid overfitting which could explain the consistent better performace over the simple Neural Network for short-term forecasting. Presumably, this gives less of an advantage in the day-ahead scenario because the data used to train is generally noisier (less information is given without the short-term lagged variables) and therefore overfitting is not a problem.

Bayesian Neural Networks also perform well in comparison, even though they tend to take a longer time to converge as they sample distributions for each parameter on each pass and back-propagate the error to the parameters of the distributions. Intuitively the same step for a classical Neural Network will take multiple steps for a Bayesian Neural Network because of the random sampling. Concrete Dropout and Functional Neural Processes also sample but back-propagate to a relatively small set of parameters controlling the distributions (for FNPs the latent distribution parameters and for Concrete Dropout the dropout rate).

The higher convergence time of Bayesian Neural Networks shows mainly in the day-ahead forecasting scenario, where the hyper-parameter optimization gave a consistently lower training epoch number for Concrete Dropout and the simple Neural Network (the Deep Ensemble is trained with the same parameters as the simple Neural Network) (see Table 5.4 and Table 5.2). This is not the case for the Bayesian Neural Network, the Functional Neural Process, and the Deep Gaussian Process. In the day-ahead scenario there is less data available, effectively causing the problem to be less complex, resulting in a lower time to converge.

In terms of training time it is also notable that the Deep Ensemble models perform as well as the simple Neural Networks, where the same parameters are used, even though an ensemble of five networks is trained. The networks are trained in parallel (as proposed in (Lakshminarayanan et al. 2017)), which results in a better usage of resources. Note that this advantage will vary based on the size of the model, the batch size, and the hardware used.

Functional Neural Processes do not perform well in terms of calibration, sharpness, and accuracy metrics. Previously we mentioned that the parameters used are likely sub-optimal due to the amount of hyper-parameters that are present in model and the increasingly high convergence times and memory usage some of them cause. Furthermore, we found that the Functional Neural Processes are very sensitive to the number of epochs trained in that they tend to become unstable with too many epochs (likely an error or numerical problem in the original code). Functional Neural Processes also ended up being slow to predict because of the sampling that is done and the graph being traversed for each sample (see Table 5.2 and Table 5.4). Concrete Dropout and the Bayesian Neural Networks that also employ sampling perform considerably better.

Overall, the methods perform similarly to some degree with an exception in Deep Gaussian Processes and Functional Neural Processes, which could likely be improved considerably given enough time. All the neural models have similar capabilities and the neural parameters we choose, like layer sizes and activation functions, are the same or similar, so they are able to perform similarly well. We find that, for different configurations and sometimes through fluctuations in the training with the same configuration, the neural models besides Functional Neural Processes perform similarly and a definite best method cannot be picked. The same applies to the evaluation of calibration and sharpness. The models are reasonably calibrated (after recalibration) and further calibration would require more representative data. The sharpness under calibration of the forecasters also fluctuated to a degree with different configurations. Sharpness represents how well the model has learnt the Aleatoric Uncertainty and for the most part correlates with how well the models perform overall. The models (besides Quantile Regression) are able to perform similarly in this regard as they are constructed similarly and learn the target function (consisting of mean and standard deviation of the target data) utilizing the same loss. It is of note that sharpness will be influenced by the Epistemic Uncertainty estimate, because the test data contains points sufficiently different to the training data. Depending on how uncertain the model gets for unseen data, the sharpness will be skewed to some degree, making the model worse by this metric. In Figure 5.7 this can be observed for the Deep Gaussian Process model.

### 6.3.2. Epistemic Uncertainty

What is left are the different ways the Epistemic Uncertainty is estimated. In Section 5.2.3 we evaluate the ability to produce Epistemic Uncertainty estimates on synthetic out of distribution (o.o.d.) data. As expected, the simple Neural Network performs poorly. The only uncertainty estimate it gives is Aleatoric Uncertainty via a model output. For the o.o.d. data the network predicts very broadly distributed values with a concentration around the area values for the training data fall in. As an Epistemic Uncertainty estimate this behavior is not useful. Quantile Regression also only provides an Aleatoric Uncertainty estimate, the quantiles predicted for the o.o.d. data are spread randomly around the quantiles for the test data and we cannot consistently conclude the Model Uncertainty from it. The other methods seem to work well. They give an uncertainty estimate aligned with the expectation that the data used for the evaluation is unfamiliar. The estimates are distinctly higher than for the test data. The randomness introduced through Dropout for Concrete Dropout and the random initialization of the ensembles for Deep Ensembles takes effect for the o.o.d. data and gives a wide range of results when sampled over. At the same time, for familiar data the randomness converges because of the training. The distributions that are used in Functional Neural Processes, Bayesian Neural Networks, and Deep Gaussian Processes can be thought of converging for observations in a similar way, while deviating strongly for data that is far from the observations. We found that Functional Neural Processes, Deep Ensembles, and Deep Gaussian Processes give good Epistemic Uncertainty estimates out of the box. Concrete Dropout and Bayesian Neural Networks require some tuning. Concrete Dropout have a model precision parameter $\tau$ (Gal et al. 2017) that influences the regularization of the learnt dropout parameter, effectively setting how much Epistemic Uncertainty the model will represent. Bayesian Neural Networks allow for setting the parameters for the prior distributions of weights and biases, influencing the Epistemic Uncertainty estimate. A problem we found with setting those parameters is that they should be tuned to accommodate for a good Epistemic Uncertainty estimation, by evaluating it in a similar way we did, otherwise it can lead to models that give as bad of an Epistemic Uncertainty estimate as the simple Neural Network. For Bayesian Neural Networks in particular the network is slower to converge with a higher prior distribution deviation and, in the simple implementation we are using, is producing very large losses to a point where the numbers can overflow. A specific pitfall in tuning these parameters is that using classical metrics, like the average error on a validation set, can lead to an overall useless model for giving an Epistemic Uncertainty estimate.

# 7. Conclusion

Extending Deep Learning with probabilistic capabilities is not a new research field but it has gained a lot of traction in recent years. Fast, reliable and good methods are still being researched, with works on improvements and new methods being released regularly. There is a lot of research done to improve performance and scalability of probabilistic extensions to Deep Learning. Probabilistic predictions are useful to interpret and evaluate model predictions, as well as to train and make design decisions for the models. In this thesis we researched and evaluated a number of methods to acquire probabilistic predictions in the domain of energy time series forecasting. The following sections sum up our findings and provide an outlook for future work.

## 7.1. Results

We find that most models have similar capabilities and assume that models like Functional Neural Processes and Deep Gaussian Processes can be brought to a similar level of quality as the other models given the right parameters. The best models beat the reference Quantile Regression and simple Neural Network models, especially in terms of the probabilistic forecast they produce.

Gaussian Processes and Deep Gaussian Processes do not scale well with large amounts of data but are often considered the baseline for probability estimates. Functional Neural Processes model similar processes with the help of a Neural Architecture but also fall short in scalability and the number of hyper-parameters to tune.

Concrete Dropout, Deep Ensembles, and Bayesian Neural Networks performed well overall, with the simpler Concrete Dropout and Deep Ensembles having an advantage in both speed and convergence time, while Bayesian Neural Networks tend to converge less well. Out of the box Functional Neural Processes, Deep Ensembles, and Deep Gaussian Processes give good Epistemic Uncertainty estimates, while Concrete Dropout and Bayesian Neural Networks have parameters that possibly need to be tuned to perform well. The Bayesian Neural Network implementation we used also ended up being more sensitive to these parameters in terms of convergence time and overall accuracy.

We explored the foundations of probabilistic Deep Learning and investigated a number of popular methods. We show that the application of the methods is feasible for energy time series forecasting and that they produce good estimates of their confidence in a forecast. Compared to more simple methods, like Quantile Regression and simple Neural Networks, they also take Epistemic Uncertainty into account and are therefore able to indicate a lack of confidence for unexpected or anomalous situations.

There will always be a measure of uncertainty in predictions that cannot be reduced, but it can be modelled, described, and managed.

## 7.2. Outlook

For continuing work in this field it would be interesting to spend more time on tuning the Functional Neural Process and the Deep Gaussian Process models, as we assume that there is still room for improvement.

We only tested Functional Neural Processes in this work, but other variants of this type of model are of interest as well. (Attentive) Neural Processes (Garnelo et al. 2018b; Kim et al. 2019) have the ability to be pre-trained with similar data to make better predictions with a small amount of actual training data. This might be of interest for time series with a scarce amount of specific data, but a decent amount of data from the same domain.

Furthermore, the application of more sophisticated forecasting strategies is of interest. The models used are, at their core, simple regressors. However there are some advanced methods that have proven to work well for temporal data, for example, recurrent methods like Long-Short-Term-Memory models (Hochreiter and Schmidhuber 1997) and the relatively new attention-based models (Vaswani et al. 2017). Neural Networks also work very well in multi-dimensional feature extraction tasks, specifically Convolutional Neural Networks (Krizhevsky et al. 2012), which have been shown to work in a forecasting context as well (Wang et al. 2017).

To improve day-ahead forecasting it is compelling to test configurations where multiple forecasts are modeled as model outputs. A Neural Network could take advantage of this sort of training by learning relationships between the time steps.

A probabilistic extension not investigated in this thesis, is the application of uncertainties as model inputs. That way additional confidence information can be used to make predictions. In the context of energy time series the incorporation of data from weather forecasts is particularly interesting. The usage of this data enables further improvement of probabilistic energy time series forecasts.

In terms of probabilistic Deep Learning in general there is a workshop at NeurIPS dedicated to Bayesian Deep Learning (`http://bayesiandeeplearning.org/`) that covers a lot of further research in the field. We gave an overview of probabilistic Deep Learning and investigated some different methods that provide confidence estimates. A lot of work is based around further research into the methods we covered, however, there are some interesting methods and variations of methods that did not fit the scope of this work. Specifically, Compound Density Networks (Kristiadi et al. 2019) and Dirichlet Prior Networks (Malinin and Gales 2018), which aim to produce Epistemic Uncertainty estimates using different approaches to what we considered in this thesis.

# Bibliography

Arora, S. and J. W. Taylor (2018). *Rule-based autoregressive moving average models for forecasting load on special days: A case study for France.* In: *European Journal of Operational Research*, Vol. 266, No. 1, pp. 259–268.

Bishop, C. M. (1994). *Mixture density networks.* In:

Blundell, C., J. Cornebise, K. Kavukcuoglu, and D. Wierstra (2015). *Weight uncertainty in neural networks.* In: *arXiv preprint arXiv:1505.05424*.

Bröcker, J. (2012). *Estimating reliability and resolution of probability forecasts through decomposition of the empirical score.* In: *Climate dynamics*, Vol. 39, No. 3-4, pp. 655–667.

Chellapilla, K., S. Puri, and P. Simard (2006). *High performance convolutional neural networks for document processing.* In:

Csáji, B. C. et al. (2001). *Approximation with artificial neural networks.* In: *Faculty of Sciences, Etvs Lornd University, Hungary*, Vol. 24, No. 48, p. 7.

Cybenko, G. (1989). *Approximation by superpositions of a sigmoidal function.* In: *Mathematics of control, signals and systems*, Vol. 2, No. 4, pp. 303–314.

Dahua, G., W. Yi, Y. Shuo, and K. Chongqing (2018). *Embedding based quantile regression neural network for probabilistic load forecasting.* In: *Journal of Modern Power Systems and Clean Energy*, Vol. 6, No. 2, pp. 244–254.

Damianou, A. and N. Lawrence (2013). *Deep gaussian processes.* In: *Artificial Intelligence and Statistics*, pp. 207–215.

Der Kiureghian, A. and O. Ditlevsen (2009). *Aleatory or epistemic? Does it matter?* In: *Structural Safety*, Vol. 31, No. 2, pp. 105–112.

Gal, Y. (2016). *Uncertainty in deep learning.* PhD thesis. PhD thesis, University of Cambridge.

Gal, Y. and Z. Ghahramani (2016). *Dropout as a bayesian approximation: Representing model uncertainty in deep learning.* In: *international conference on machine learning*, pp. 1050–1059.

Gal, Y., J. Hron, and A. Kendall (2017). *Concrete dropout.* In: *Advances in Neural Information Processing Systems*, pp. 3581–3590.

Gardner, J. R., G. Pleiss, D. Bindel, K. Q. Weinberger, and A. G. Wilson (2018). *GPyTorch: Blackbox Matrix-Matrix Gaussian Process Inference with GPU Acceleration.* In: *Advances in Neural Information Processing Systems.*

Garnelo, M., D. Rosenbaum, C. J. Maddison, T. Ramalho, D. Saxton, M. Shanahan, Y. W. Teh, D. J. Rezende, and S. Eslami (2018a). *Conditional neural processes.* In: *arXiv preprint arXiv:1807.01613.*

Garnelo, M., J. Schwarz, D. Rosenbaum, F. Viola, D. J. Rezende, S. Eslami, and Y. W. Teh (2018b). *Neural processes.* In: *arXiv preprint arXiv:1807.01622.*

Gneiting, T., F. Balabdaoui, and A. E. Raftery (2007). *Probabilistic forecasts, calibration and sharpness.* In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, Vol. 69, No. 2, pp. 243–268.

Gneiting, T., A. E. Raftery, A. H. Westveld III, and T. Goldman (2005). *Calibrated probabilistic forecasting using ensemble model output statistics and minimum CRPS estimation.* In: *Monthly Weather Review*, Vol. 133, No. 5, pp. 1098–1118.

Goodfellow, I., J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio (2014). *Generative adversarial nets.* In: *Advances in neural information processing systems*, pp. 2672–2680.

Hamill, T. M. (2001). *Interpretation of rank histograms for verifying ensemble forecasts.* In: *Monthly Weather Review*, Vol. 129, No. 3, pp. 550–560.

Hammer, B. and K. Gersmann (2003). *A note on the universal approximation capability of support vector machines.* In: *neural processing letters*, Vol. 17, No. 1, pp. 43–53.

Head, T. et al. (Mar. 2018). *scikit-optimize/scikit-optimize: v0.5.2.* Version v0.5.2. URL: `https://doi.org/10.5281/zenodo.1207017`.

Hinton, G. E., N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov (2012). *Improving neural networks by preventing co-adaptation of feature detectors.* In: *arXiv preprint arXiv:1207.0580.*

Hochreiter, S. and J. Schmidhuber (1997). *Long short-term memory.* In: *Neural computation*, Vol. 9, No. 8, pp. 1735–1780.

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

Hornik, K. (1991). *Approximation capabilities of multilayer feedforward networks.* In: *Neural networks*, Vol. 4, No. 2, pp. 251–257.

Kendall, A. and Y. Gal (2017). *What uncertainties do we need in bayesian deep learning for computer vision?* In: *Advances in neural information processing systems*, pp. 5574–5584.

Kendall, A., Y. Gal, and R. Cipolla (2018). *Multi-task learning using uncertainty to weigh losses for scene geometry and semantics.* In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 7482–7491.

Kim, H. (2019). *Bayesian Neural Network For Pytorch.* `https://github.com/Harry24k/bayesian-neural-network-pytorch`.

Kim, H., A. Mnih, J. Schwarz, M. Garnelo, A. Eslami, D. Rosenbaum, O. Vinyals, and Y. W. Teh (2019). *Attentive neural processes.* In: *arXiv preprint arXiv:1901.05761.*

Koenker, R. and K. F. Hallock (2001). *Quantile regression.* In: *Journal of economic perspectives*, Vol. 15, No. 4, pp. 143–156.

Kristiadi, A., S. Däubener, and A. Fischer (2019). *Predictive uncertainty quantification with compound density networks.* In: *arXiv preprint arXiv:1902.01080.*

Krizhevsky, A., I. Sutskever, and G. E. Hinton (2012). *Imagenet classification with deep convolutional neural networks.* In: *Advances in neural information processing systems*, pp. 1097–1105.

Lakshminarayanan, B., A. Pritzel, and C. Blundell (2017). *Simple and scalable predictive uncertainty estimation using deep ensembles.* In: *Advances in Neural Information Processing Systems*, pp. 6402–6413.

LeCun, Y., Y. Bengio, and G. Hinton (2015). *Deep learning.* In: *nature*, Vol. 521, No. 7553, pp. 436–444.

Louizos, C., X. Shi, K. Schutte, and M. Welling (2019). *The Functional Neural Process.* In: *Advances in Neural Information Processing Systems.*

MacKay, D. J. (1992). *Bayesian interpolation.* In: *Neural computation*, Vol. 4, No. 3, pp. 415–447.

Malinin, A. and M. Gales (2018). *Predictive uncertainty estimation via prior networks.* In: *Advances in Neural Information Processing Systems*, pp. 7047–7058.

Melis, G., C. Blundell, T. Kočiskỳ, K. M. Hermann, C. Dyer, and P. Blunsom (2018). *Pushing the bounds of dropout.* In: *arXiv preprint arXiv:1805.09208.*

Neal, R. M. (2012). *Bayesian learning for neural networks.* Vol. 118. Springer Science & Business Media.

Nix, D. A. and A. S. Weigend (1994). *Estimating the mean and variance of the target probability distribution.* In: *Proceedings of 1994 IEEE International Conference on Neural Networks (ICNN'94).* Vol. 1. IEEE, pp. 55–60.

Oh, K.-S. and K. Jung (2004). *GPU implementation of neural networks.* In: *Pattern Recognition*, Vol. 37, No. 6, pp. 1311–1314.

*Open Power System Data. 2019. Data Package Time series. Version 2019-06-05* (2019). Primary data from various sources, for a complete list see URL. URL: `https://doi.org/10.25832/time_series/2019-06-05`.

Paszke, A., S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala (2019). *PyTorch: An Imperative Style, High-Performance Deep Learning Library.* In: *Advances in Neural Information Processing Systems 32.* Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett. Curran Associates, Inc., pp. 8024–8035. URL: `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

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

Salinas, D., V. Flunkert, and J. Gasthaus (2017). *DeepAR: Probabilistic forecasting with autoregressive recurrent networks.* In: *arXiv preprint arXiv:1704.04110.*

Scarselli, F., M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini (2008). *The graph neural network model.* In: *IEEE Transactions on Neural Networks*, Vol. 20, No. 1, pp. 61–80.

Seedat, N. and C. Kanan (2019). *Towards calibrated and scalable uncertainty representations for neural networks.* In: *arXiv preprint arXiv:1911.00104.*

Shridhar, K., F. Laumann, and M. Liwicki (2018). *Uncertainty Estimations by Softplus normalization in Bayesian Convolutional Neural Networks with Variational Inference.* In: *arXiv preprint arXiv:1806.05978.*

Srivastava, N., G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov (2014). *Dropout: a simple way to prevent neural networks from overfitting.* In: *The journal of machine learning research*, Vol. 15, No. 1, pp. 1929–1958.

Szegedy, C., W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus (2013). *Intriguing properties of neural networks.* In: *arXiv preprint arXiv:1312.6199.*

Thorarinsdottir, T. L., M. Scheuerer, and C. Heinz (2016). *Assessing the calibration of high-dimensional ensemble forecasts using rank histograms.* In: *Journal of computational and graphical statistics*, Vol. 25, No. 1, pp. 105–122.

Tsyplakov, A. (2013). *Evaluation of probabilistic forecasts: proper scoring rules and moments.* In: *Available at SSRN 2236605.*

Vaswani, A., N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin (2017). *Attention is all you need.* In: *Advances in neural information processing systems*, pp. 5998–6008.

Wang, H. ZHI, G. QIANG Li, G. BIN Wang, J. CHUN Peng, H. Jiang, and Y. TAO Liu (2017). *Deep learning based ensemble approach for probabilistic wind power forecasting.* In: *Applied Energy*, Vol. 188, pp. 56 –70. URL: `http://www.sciencedirect.com/science/article/pii/S0306261916317421`.

Williams, C. K. and C. E. Rasmussen (2006). *Gaussian processes for machine learning.* Vol. 2. 3. MIT press Cambridge, MA. Chap. 2.

Wilson, A. G., Z. Hu, R. Salakhutdinov, and E. P. Xing (2016). *Deep kernel learning.* In: *Artificial Intelligence and Statistics*, pp. 370–378.

Zhang, G. P. and M. Qi (2005). *Neural network forecasting for seasonal and trend time series.* In: *European journal of operational research*, Vol. 160, No. 2, pp. 501–514.

Zhou, Z.-H. and J. Feng (2017). *Deep forest.* In: *arXiv preprint arXiv:1702.08835.*

Zhu, L. and N. Laptev (2017). *Deep and confident prediction for time series at uber.* In: *2017 IEEE International Conference on Data Mining Workshops (ICDMW).* IEEE, pp. 103–110.

# A. Appendix

Following is some additional technical information on the thesis and its implementation.

## A.1. Implementation

To implement the processes and methods needed for this thesis we used Python. As for the Deep Learning framework we chose PyTorch (Paszke et al. 2019). Gaussian Processes are implemented with help of GPyTorch (Gardner et al. 2018), and Bayesian Neural Networks with a simple Bayesian Neural Network library (Kim 2019), both based on PyTorch. Code for Concrete Dropout and Functional Neural Processes is sourced and adapted from the original authors of the publications (Gal et al. 2017; Louizos et al. 2019). Deep Ensembles are implemented according to the specifications from the paper Lakshminarayanan et al. (2017).

The models are wrapped for convenient prediction and hyper-parameter optimization with scikit-learn (Pedregosa et al. 2011) and scikit-optimize (Head et al. 2018) for Bayesian Optimization.

The project code implemented for this thesis, along with instructions on how to use it, can be found at:
`https://github.com/yannick-t/probabilistic_forecasting_of_energy_time_series_using_deep_learning`.[1] The specific versions of all the dependencies is also located there as an environment description file for anaconda.

[1] At the time of writing, the latest commit is 7fc960729c4ce21cbebb4e68d9fdece08abfbeb0. Direct link: `https://github.com/yannick-t/probabilistic_forecasting_of_energy_time_series_using_deep_learning/tree/7fc960729c4ce21cbebb4e68d9fdece08abfbeb0`.

## A.2. Feature Construction

For the load forecasting scenario we empirically constructed a number of features that we list in detail here:

- The day of year, transformed with a sinus function.
- The day of year, transformed with a cosinus function as differentiation / trend of the sinus transformed version.
- The hour of the day, transformed with a sinus function.
- The hour of the day, transformed with a cosinus function as differentiation / trend of the sinus transformed version.
- The raw time of day.
- The day of the week, one-hot encoded.
- For all considered calendars: (for Germany: whole of Germany and some high load states)
  - Special days (holidays), one-hot encoded.
  - Special day bridging days (Mondays / Fridays before and after special days on weekdays), one-hot encoded.
  - Special day double bridging days (Mondays / Fridays / Thursdays / Tuesdays before and after special days on weekdays), one-hot encoded.
- Indicator variable for christmas time (christmas till new year's).
- Indicator variable for new year's time (early time in the January that is usually anomalous).
- Lagged variables:
  - Load for same time 1, 2, 3, 4, 5 and 7 days before.
  - Load for 15, 30, 45, 60, 120, 180 minutes before (only for short term forecasting).

## A.3. Hyper-Parameters

We optimized the hyper-parameters of the models using Bayesian Optimization and three-fold cross-validation. The parameters that were optimized were optimized for the two forecasting scenarios (with and without short term lagged variables) separately.

**Table A.1.** Table of general Hyper-parameters used for the short-term load forecasting scenario. For the Functional Neural Process the layer sizes referenced are for the Encoder (Enc.) and Decoder (Dec.).

| | Simple NN | Concrete | Deep Ens. | BNN | Deep GP | FNP |
|---|---|---|---|---|---|---|
| Layer Sizes | 24, 64, 32 | 24, 64, 32 | 24, 64, 32 | 24, 64, 32 | 4 | Enc. 24, 64<br>Dec. 32 |
| Epochs | 3500 | 3400 | 3500 | 3306 | 600 | 300 |
| Learning Rate | 0.0005 | 0.00051 | 0.0005 | 0.000182 | 0.0015 | 0.001 |
| Batch Size | 1024 | 1024 | 1024 | 1024 | 1024 | 1024 |

**Table A.2.** Table of general hyper-parameters used for the day-ahead load forecasting scenario. For the Functional Neural Process the layer sizes referenced are for the Encoder (Enc.) and Decoder (Dec.).

| | Simple NN | Concrete | Deep Ens. | BNN | Deep GP | FNP |
|---|---|---|---|---|---|---|
| Layer Sizes | 132, 77, 50 | 132, 77, 50 | 132, 77, 50 | 132, 77, 50 | 4 | Enc. 132, 77<br>Dec. 50 |
| Epochs | 1253 | 271 | 1253 | 3500 | 600 | 300 |
| Learning Rate | 0.00005 | 0.0001 | 0.0000503 | 0.000423 | 0.0015 | 0.001 |
| Batch Size | 1024 | 1024 | 1024 | 1024 | 1024 | 1024 |

We could not feasibly optimize all parameters of all methods together so we used best practices and good values from previous tests, as well as optimizing some parameters separately. Generally we use a batch size of 1024,n a ReLu activation function for all layers but the last (for Neural Network models) and the Adam optimizer. The number of units for three layers was optimized for a simple reference Neural Network and used for Concrete Dropout, Deep Ensemble and Bayesian Neural Network models, with similar values used for the encoder and decoder of the Functional Neural Processes. The number of epochs and the learning rate were optimized together extensively for the reference Neural Network model, Concrete Dropout and Bayesian Neural Networks with an upper limit of 3500 epochs (4200 for Bayesian Neural Networks as they are slower to converge).

For the Deep Ensembles the same parameters are used as the reference Neural Network model. Furthermore we use an ensemble size of 5 for the Deep Ensembles as proposed in the original paper (Lakshminarayanan et al. 2017).

To keep the optimization time practical we chose feasible upper limits for the optimizations, in particular the hidden layer size for Deep Gaussian Processes and the reference set size for Functional Neural Processes had to be kept relatively small.

**Table A.3.** Table of Functional Neural Process specific hyper-parameters that are not in the other tables used for both load forecasting scenarios. The reference set size is a percentage of the training data-set size.

| Short-Term | U Dim. | Z Dim. | Free Bits Z | Reference Set Size [%] |
|---|---|---|---|---|
| ✓ | 2 | 32 | 1.0 | 0.3 |
| ✗ | 9 | 32 | 1.0 | 0.3 |

In Table A.1, Table A.2 and Table A.3 the important parameters used for this work are listed, for the exact configurations and code refer to the repository[2]. The Concrete Dropout models have some extra hyper-parameters, namely model precision $\tau$ and length-scale. These parameters are used to calculate regularization parameters for the model (Gal et al. 2017). $\tau$ has influence over the regularization of the dropout, meaning, the Epistemic Uncertainty estimate. For both scenarios we used $1 \cdot 10^{-4}$ for length-scale and $2 \cdot 10^{-3}$ as $\tau$. As prior distribution parameters for the Bayesian Neural Network models we use a $\mu$ of 0.0 and a $\sigma$ of 0.1 for the short-term scenario and a $\mu$ of 0.0 and a $\sigma$ of 0.161 for the day-ahead scenario, which proved to work well and give good Epistemic Uncertainty estimates.

[2] `https://github.com/yannick-t/probabilistic_forecasting_of_energy_time_series_using_deep_learning`