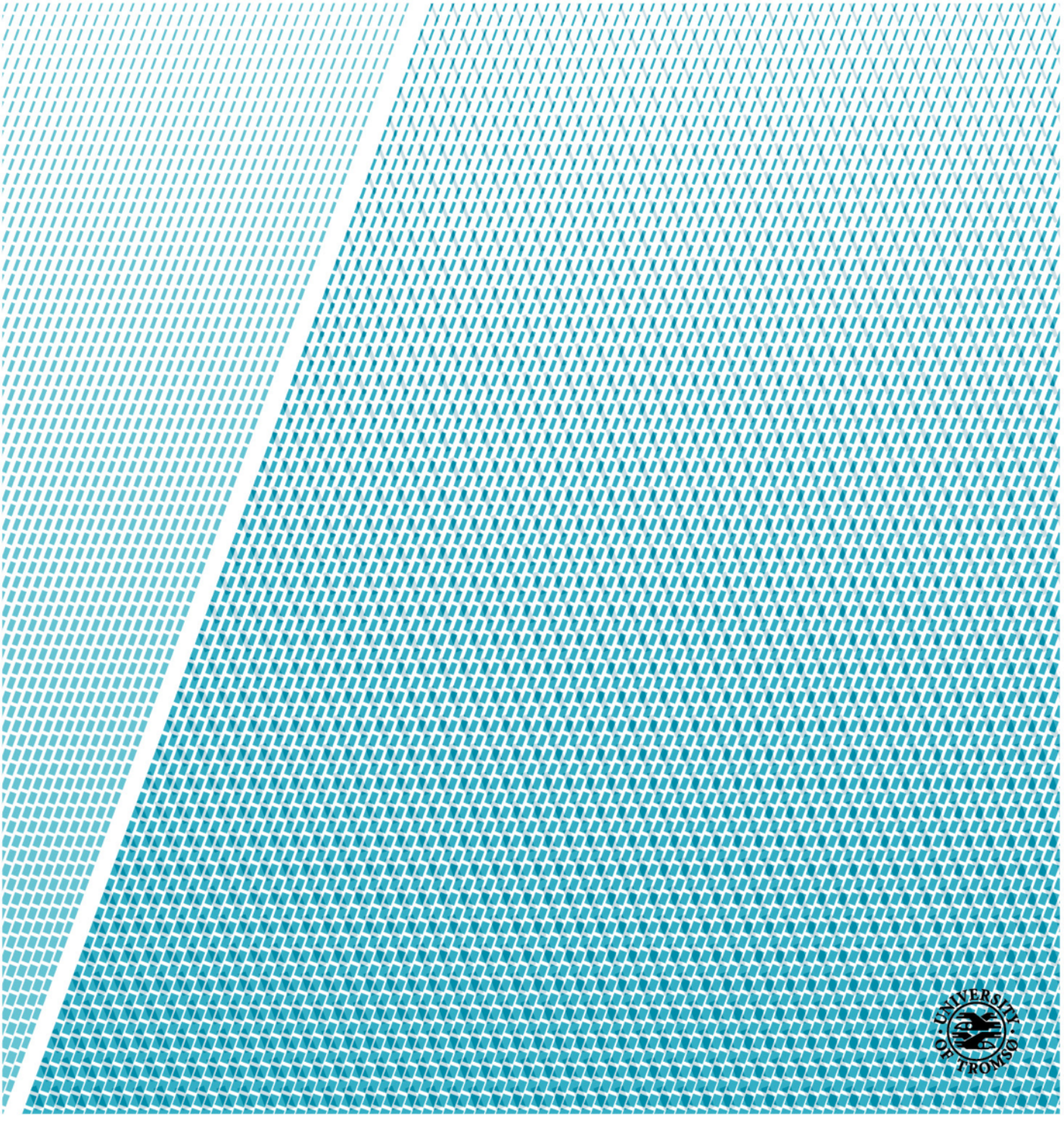
UiT
THE ARCTIC
UNIVERSITY
OF NORWAY

Faculty of Science and Technology, Department of Physics and Technology

Probabilistic Wind Power Forecasting with Deep Neural Sequence Models

Sofie Svenøe

EOM-3901 Master's Thesis in Energy, Climate and Environment 30 SP - June 2023



Abstract

As the world strives to fulfill the goal of zero-emission established during the Paris agreement (UN, 2015), an increasing amount of wind power is integrated into the liberalized electricity markets. With this escalation comes the need for wind power forecasting (WPF) due to the intermittent nature of wind, and WPF is therefore becoming an important field of study to successfully incorporate wind power to the electricity market (Wang, Zou, Liu, Zhang, & Liu, 2021). Given the rapid growth of machine learning, deep learning and probabilistic forecasting has emerged as good alternatives for WPF (Eikeland, Hovem, Olsen, Chiesa, & Bianchi, 2022) due to their non-linear processing methods and their ability to model uncertainties.

In this study, two probabilistic deep learning networks and a statistical model are tested as WPF models for a 54 MW wind power park. The models are trained to predict for the day-ahead and intraday electricity market, which respectively has 12-26 h and 1-24 h as associated forecasting horizons. Historical wind power production and Numerical weather predictions (NWP) are used as input to the WPF models. NWPs are modeled from the MEPS model, operated by the Norwegian Meteorology Institute (MET Norway).

The tests show that the two neural network models Temporal Fusion Transformer, and DeepAR, produces better predictions than the statistical model, SARIMAX, for the day-ahead market. The neural networks achieved P50/P90-Risk respectively of 0.153/0.081, and 0.175/0.091. While, for the intraday market, the models DeepAR, and SARIMAX performed substantially better than the Temporal Fusion Transformer, with P50/P90-Risk of respectively, 0.111/0.056, and 0.184/0.099. This implies that Transformer sequence models perform best on long-term forecasting, whereas autoregressive models still perform best on short-term forecasting.

Acknowledgments

First of all, I would like to thank the excellent group of Italians that have supervised me on this thesis. Thank you Filippo Maria Bianchi, Michele Guerra, and Matteo Chiesa for your encouragement, guidance and good spirits. Additionally, thank you Filippo and Matteo for bringing me on this 2 year journey of deep learning.

I would also like to thank my classmates for some great years spent in Tromsø, and my friends in Trondheim for being the best company to write the thesis with. Thank you Vortex NTNU, for giving me a place to write when I had no place to go.

Lastly, to my family, and Kristian: thank you for your love, support, and patience during this year.

Sofie Svenøe,
Trondheim, June 2023.

Table of Contents

List of Tables

List of Figures

Abbreviations

ACF	Autocorrelation Function
Adam	Adaptive Moment Estimation
AI	Artificial Intelligence
ANN	Artificial Neural Network
AR	Autoregressive
ARIMA	Autoregressive Integrated Moving Average
ARMA	Autoregressive Moving Average
BPTT	Backpropagation Through Time
CDF	Cumulative Density Function
CRPS	Continuous Ranked Probability Score
DL	Deep Learning
GLU	Gated Linear Unit
GRN	Gated Residual Network
GRU	Gated Recurrent Unit
LSTM	Long Short-Term Memory
MA	Moving Average
MEPS	MetCoOp Ensemble Prediction System
MIMO	Multiple Input, Multiple Output
ML	Machine Learning
NLP	Natural Language Processing
NWP	Numerical Weather Prediction
PACF	Partial Autocorrelation Function
PDF	Probability Density Function
PI	Prediction Interval
p-Risk	Percentile Risk
qCRPS	Quantile weighted CRPS
QL	Quantile Loss
QR	Quantile Regression
RNN	Recurrent Neural Network
SARIMA	Seasonal Autoregressive Integrated Moving Average
SARIMAX	SARIMA with eXogenous input
seq2seq	Sequence to Sequence
SGD	Stochastic Gradient Descent
TFT	Temporal Fusion Transformer
WPF	Wind Power Forecasting

Part I / Introduction

1 Motivation

Electricity exchanges are today mostly done on liberalized electricity markets such as Nord Pool. In 2014, 84% of the electricity exchanges in the European Nordic was done through Nord Pool (Mazzi & Pinson, 2017). Exchanges on such markets take the form of future contracts and auctions, where the majority of the exchanges takes place in a day-ahead auction. This form of electricity exchange came forth at a time when power production consisted mostly of plants that had the flexibility to turn on and off and plan their production ahead of production time. In the present day, with an increased share of volatile renewable power in the power pool, day-ahead power production has become less predictable due to being more dependent on intermittent weather conditions. Another option is to exchange at the intraday auctions, which takes place continuously during the same day as production. In this auction, more accurate bids can be delivered due to being closer to production time. With the electricity market of today, producers risk facing financial penalties if there are mismatches between the agreed-upon generation and the amount of power delivered (Mazzi & Pinson, 2017). Volatile renewable power producers, therefore, have a larger risk for financial penalties and must operate with a trade-off between maximizing power production or minimizing risk.

Wind power is such an energy source that is characterized by volatile power production and its integration into the electricity market therefore relies on forecasting models that can give good predictions about future power production. Wind power forecasting (WPF) refers to forecasting the expected wind power production either regionally or locally and has been a field of research ever since the introduction of wind power in the late 80s (Giebel & Kariniotakis, 2017). Being highly dependent on weather features, the field of wind power forecasting is closely related to the field of meteorology and weather predictions. Surrounding topography adds complexity to the issue as conditions can vary significantly from one wind park to another, meaning that no one WPF model can be successfully applied to every wind park. Alas, wind power forecasting is a diversified issue and is therefore a multidisciplinary research area that encompasses not only energy analytics but also applied mathematics, artificial intelligence, software engineering, and other fields.

Until recently, point forecasting models predicting the most likely value have been the dominating forecasting method (Cini, Marisca, Bianchi, & Alippi, 2023). Predicting wind power entails forecasting a time series with very short temporal dependencies in contrast to what you would typically see in e.g. electrical consumption or traffic time series. However, a wind power production time series does not only depend on its previous values but also heavily depends on wind, humidity, and surrounding topography. These dependencies are of a complex and highly non-linear nature, making accurate point predictions almost impossible (Eikeland et al., 2022). It is therefore necessary to use forecasting models that can model the uncertainties in their output, and such models are termed *probabilistic*. Probabilistic forecasting models provide additional information about the uncertainty of the most likely value, which can assist the wind power trader in making informed

decisions. Probabilistic forecasting models give predictions for each point as a range of values, usually as confidence intervals or density functions (Fornacon-Wood et al., 2022), and uncertainty is therefore reflected in the width of the distribution.

Time series forecasting is a sequence processing problem that historically has been solved by statistical models and has earned its popularity due to being easy to use and interpret. However, most statistical methods are limited by linear assumptions, making them inappropriate to use on complex tasks such as wind power forecasting. Therefore, deep learning, and especially neural sequence models, has become a prominent research field within wind power forecasting due to its capabilities of mapping non-linear dependencies. Neural sequence processing is primarily not limited by such linear assumptions as statistical models, however, they are generally considered as difficult to interpret and has therefore commonly been called black boxes. This is due to deep learning models having the advantage of working reasonably well without the need of having knowledge about the distribution of the data and generally requiring less preprocessing than statistical models (Gasthaus et al., 2019). Nevertheless, having knowledge about the nature of the data can significantly reduce the training effort and produce even better results.

As neural networks are difficult to implement, especially for non-machine learning experts, this thesis aims to develop and test deep learning models for wind power forecasting that can be ready to use for real-world practitioners. The goal is to test if the models are ready to use without the need of software engineering and only needing to tune and train the model on the given park, such that wind power producers can use the models practically 'out of the box'.

2 Research Questions, Proposed Approach, and Contributions

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

- *How does the accuracy and uncertainty of probabilistic forecasts generated from deep sequential models compare between forecasts for the day-ahead and intraday markets?*
- *To what extent can wind power forecasts be improved using deep sequential models, rather than the previously popular statistical models?*

The proposed approach of this thesis is to evaluate and compare various probabilistic forecasting methods for the day-ahead and intraday markets, with an emphasis on evaluating their performance in terms of prediction accuracy and uncertainty. The investigation will analyze the forecasts within their respective market time frames, where it is acknowledged that the day-ahead forecast carries a higher level of uncertainty compared to the intraday forecast.

The objective is to provide forecasts where the wind power trader can take advantage of the uncertainty in the day-ahead forecasts to determine a lower bound of production,

and submit safe bids to the day-ahead market. Additionally, the intraday forecast aims to deliver accurate predictions that can adequately adjust the day-ahead bids in the intraday market. The ambition is that the forecasts can be used to minimize discrepancies between the agreed-upon generation and the actual power delivered by providing reliable and accurate forecasts.

This master thesis is a contribution to the research field on wind power forecasting at UiT and focuses on Fakken wind power park. The key contributions can be summarized as follows:

- Proposing a dual bidding situation to increase accuracy using forecasts both for the day-ahead and intraday market.
- Testing the methods using multivariate wind power datasets from Fakken wind power park using numerical weather predictions as features.
- Investigating if the models are ready for usage by non-machine learning experts.

3 Thesis Outline

This thesis is divided into five parts, and this section concludes the introduction. Part II contains relevant technical background information needed to understand the concepts dealt with in this thesis. The topics covered in the technical background include introductions to *time series forecasting*, *neural network training*, *sequential neural networks*, and *wind power forecasting for the electricity market*.

Part III presents the proposed method and the sequential models used in this thesis.

The experiments conducted and the results thereof are presented in part IV.

Finally, in part V, a summary along with concluding remarks and instructions for further work is presented.

Part II / Technical Background

4 Time Series Forecasting

A time series is by (Kirchgässner, Wolters, & Hassler, 2012) defined as a set of quantitative observations arranged in chronological order. When ordering the observation by time of record it becomes possible to identify and extract knowledge of the underlying process. Two mature areas in the field of time series are time series analysis and time series forecasting, where the first revolves around extracting knowledge in the form of seasonality and trends while the latter revolves around using such knowledge to create mathematical models to predict future values of the time series.

A time series forecasting model, $f(\cdot)$, aims to model the future values of a time series y given past values of the time series and exogenous variables $\mathbf{x}$

$$\hat{y}_{(t+1):(t+T)} = f(y_{(t-\tau):t}, \mathbf{x}_{(t-\tau):(t+T)}), \tag{1}$$

where the predicted values from the forecasting model are denoted as $\hat{y}$, while the exogenous variables can include both past and future values. Past values are limited to a look-back window, τ, and future values are restricted to a forecast horizon denoted as T.

The next sections will describe the framework and theory behind setting up a time series forecasting problems, and the basics behind probabilistic forecasting.

4.1 Single- and Multi-Step Forecasting

One of the first decisions to make when setting up the forecasting problem is if the model should output single- or multi-step forecasts, where the former makes one-step-ahead forecasts and the latter makes forecasts for multiple time steps ahead.

Two common methods for computing multi-step forecasts are the *recursive* and *Multiple-Input-Multiple-Output (MIMO)* methods (Taieb, Bontempi, Atiya, & Sorjamaa, 2012). The recursive strategy involves recursively feeding the output from a single-step forecasting model until the T-step prediction has been made. The MIMO strategy is a more direct method, which in one process outputs a vector of values instead of a scalar.

It is generally intuitive given the forecasting problem whether to forecast one or multiple time steps ahead. However, certain considerations must be taken into account when performing multi-step forecasting. Due to the longer forecasting horizon, each predicted value has a higher level of uncertainty. Especially the recursive method is prone to error accumulation due to using the previous predicted value as input for the next time step. Multi-step forecasting methods are therefore said to have lower accuracy than single-step forecasting methods.

4.2 The role of Covariates

A time series that includes multiple variables is called a multivariate time series. In contrast, a time series with only one variable is referred to as a univariate time series. The variables in a multivariate time series are classified either as endogenous or exogenous. Endogenous variables are influenced by other variables within the system, while exogenous variables, also called *covariates*, are not. It is important to note that an exogenous variable should have an effect on the output variable, otherwise, there is no reason to include it in the dataset (Brownlee, 2018).

In time series forecasting it is typical to work with multivariate time series. This is because time series forecasting often has a lot of data associated with the same problem. Therefore a time series forecasting model has a higher chance of getting good results when using covariates since the model can learn dependencies between different time series. For example, a time series with electric load from households will have a strong relationship to the corresponding outside temperature. Similarly, a traffic load time series will be highly dependent on the time of day and day of the week. Some time series are almost infeasible to forecast without the use of covariates. A typical example is weather-related time series like wind and solar power production, or stock market time series like Nasdaq closing prices.

There exist several ways that covariates can be included in a time series forecasting model and the general distinction is if the covariates are known or unknown in the future. An example of unknown covariates can be measured weather features, while known covariates can be predicted weather features or date-time features like a day of week or week of the year. Another type of covariate is static in time and is usually location-based. E.g. retail time series from different stores selling the same items.

Additionally, the use of covariates makes it possible to have smaller datasets. A model training on univariate data needs a much larger training dataset to make good predictions during inference due to only learning inter-series temporal dependencies.

4.3 Time Series Forecasting as a Supervised Learning Problem

A time series is composed of a set of sequences with length depending on the dataset and the forecasting problem. For example, a forecasting model trained to predict day-ahead values can have input sequences of different lengths depending on the long-term temporal dependencies seen in the dataset. This trait is often called the memory of the time series and describes how long information is detained in future values. For example, a time series containing values dependent on the day of the week might have a memory as long as 1-2 weeks, while a time series of seemingly random values will have a much shorter memory.

The concept of time series memory gives the setup of time series forecasting as a supervised learning problem where input-output pairs are created from the time series. Supervised

learning in itself involves training the forecasting model on labeled data, which means that the model learns to map input data to the corresponding output data. The objective of the learning is to minimize the difference, or *error*, between the predicted values and labels.

To simplify notations, $\mathbf{z_{t-\tau,t}}$ will be used for the input pairs. When the time range lies before prediction time $[t-\tau, t-1]$, both endogenous, and exogenous variables will be passed as input $\mathbf{z_t} = (y_t, \mathbf{x_t})$. While for time ranges after prediction time, $[t, t+T]$, only exogenous variables $\mathbf{z_t} = \mathbf{x_t}$ will be passed as input. An input-output pair to the forecasting model will have the general form:

Input sequence: $[\mathbf{z_{t-\tau}}, ..., \mathbf{z_{t-1}}, \mathbf{z_t}, ..., \mathbf{z_{t+T}}] = Z_{t-\tau,t+T}$
Label sequence: $[y_{t+1}, y_{t+2}, ..., y_{t+T}] = Y_{(t+1):(t+T)}$

where τ represents the *memory* or *look-back window* and T represents the *forecast horizon*.

This type of dataset configuration is called a *sliding* or *rolling window dataset* and gets its name from the procedure of "sliding" a window of fixed length across the time series to create samples for the forecasting model. The step size of the window can be 1 or more, meaning that the next sample is incremented by 1 or multiple time steps.

4.4 Probabilistic Forecasting

Probabilistic forecasting models give information about uncertainty for each predicted time step, usually in the form of prediction intervals or probability density functions (pdfs). Such models differ from the historically most common models which were point forecasters, meaning that for each time step they predicted one point - the most likely value. This works well for deep learning tasks as machine translation, sentiment analysis etc, but for planning purposes, it gives the practitioner little information about the future. Probabilistic forecasting has therefore received popularity in time series forecasting the recent years and especially the energy trader will be able to give more effective bids while having a good understanding of the risks (Nowotarski & Weron, 2018).

Probabilistic forecasting can generally be grouped into frequentist and Bayesian inference. The difference lies in how uncertainty is assigned. Frequentist approaches assigns probabilities to the observed data. Bayesian approaches build on Bayes' theorem and assign probabilities to hypotheses based on the observed data (Fornacon-Wood et al., 2022).

4.4.1 Frequentist Inference

There exist several approaches to frequentist probabilistic forecasting (Jensen, Bianchi, & Anfinsen, 2022). The basic technique is prediction intervals (PIs), where a confidence interval sets the lower and upper bounds with the point forecast set as the center of the interval (Nowotarski & Weron, 2018). A 95% confidence interval is valid if 95% of the

observations lie inside the PI.
The generic form for computing the upper and lower bound of PIs is

$$P(Y_t \in C(Z_t)) \leq 1 - \alpha, \tag{2}$$

where Y_t is the response variable, $C(Z_t)$ is the confidence interval centered at the point forecast and α is the significance level.

It is often beneficial to construct several PIs, and in this case, the technique is called quantile regression (QR). With the predicted point forecasts, QR can be applied to the input set Z_t with (Nowotarski & Weron, 2018)

$$Q_{P_t}(q|Z_t) = Z_t\beta_q, \tag{3}$$

where $Q_{P_t}(q|\cdot)$ is the conditional q-th quantile of the target distribution, X_t is the regressors, often point forecasts, and β_q is a vector of parameters for quantile q. The parameters are estimations found by minimizing the quantile/pinball loss function

$$\min_{\beta_q} \left[\sum_{\{t:P_t \geq Z_t, \beta_q\}} q \cdot |P_t - Z_t\beta_q| + \sum_{\{t:P_t < Z_t, \beta_q\}} (1-q) \cdot |P_t - Z_t\beta_q| \right]. \tag{4}$$

The quantile loss function is a strictly proper scoring rule especially designed for penalizing quantile forecasts (Gneiting, 2011). The error calculated by quantile loss measures the distance between the true value and each quantile prediction, and thereafter takes into account whether the model underestimates or overestimates. A large quantile will be more penalized for underpredicting than a low quantile. Similarly, a low quantile will be more penalized for overpredicting than a large quantile. It is common to come across the quantile function with simpler notations using predicted values $\hat{y}$ and labels y for one time step

$$QL(y, \hat{y}, q) = q(y - \hat{y})_+ + (1-q)(\hat{y} - y)_+, \tag{5}$$

where over and under-prediction are distinguished by using the max function $(\cdot)_+ = max(0, \cdot)$.

4.4.2 Bayesian Inference

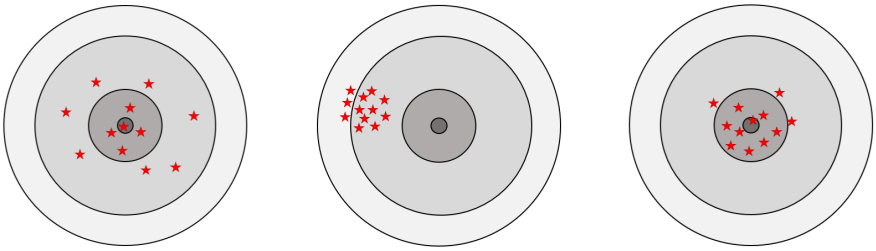
Bayesian inference depends on estimating the underlying pdf from the available data. This can either be done by estimating the pdf given known parameters or assuming a pdf and estimating the parameters (Theodoridis & Koutroumbas, 2009). Another common term for Bayesian inference is therefore parametric inference. The objective is to estimate the conditional distribution of the future data given the past data and the covariates

$$P(\mathbf{y}_{t:t+T}|\mathbf{z}_{t-\tau:t+T}). \tag{6}$$

However, it is in most cases infeasible to directly estimate eq. 6 and it is further assumed that this distribution consists of a product of likelihood factors

$$P_\theta(\mathbf{y}_{t:t+T}|\mathbf{z}_{t-\tau:t+T}) = \prod_{t'=t-\tau}^{t+T} P_\theta(\mathbf{y}_{t'}|\mathbf{z}_{t'}) = \prod_{t'=t-\tau}^{t+T} \mathcal{L}(\mathbf{z}_{\mathbf{t}'}|\theta), \tag{7}$$

where θ are parameters of the distribution typically obtained by the predictive model. The problem then becomes generating a set of parameters to estimate the likelihood for each time step. The most common method is to assume Gaussian likelihoods, which often is due to mathematical simplicity instead of empirical evidence.



(a) calibrated, not sharp

(b) sharp, not calibrated

(c) sharp and calibrated

Figure 1: Example of sharpness and calibration

4.4.3 Assessing the quality of probabilistic forecasts

The quality of a probabilistic forecast is characterized by *sharpness* and *calibration*. As seen in Fig. 1, sharpness and calibration can intuitively be described by shots on a shooting target. Sharpness refers to the density of the shots, i.e. if they are located close to each other. Calibration refers to the accuracy of the shots, and how many are hits or misses. For probabilistic forecasts, sharpness refers to the width and tightness of the PI while calibration refers to the accuracy of the PI. An 80 % PI should have 80% of the observed values inside the PI. A sharp and calibrated forecast is therefore one which has compact PIs and meets the required accuracy.

4.5 Time Series Forecasting Models

Statistical models have, until recently, dominated the research on time series forecasting (Dang-Ha, Bianchi, & Olsson, 2017). The methods are mostly state space models (SSM), which are usually univariate parametric models where the parameters must be decided by the practitioner. The most influential frameworks are Exponential Smoothing (R. Hyndman, Koehler, Ord, & Snyder, 2008), Box and Jenkins autoregressive integrated moving average (ARIMA) (Jenkins, 1970), and its multivariate extensions, ARIMAX (De Gooijer & Hyndman, 2006). Common to many statistical models is that they are limited to modeling linear dependencies, while also falling short when modeling long-term dependencies (Mashlakov, Kuronen, Lensu, Kaarna, & Honkapuro, 2021). The research has therefore trended towards deep learning (DL) models, which due to their neural architecture are better suited to model non-linear dependencies. Nevertheless, due to its historical popularity, the ARIMA model will also be tested in this thesis. Due to its complexity, neural sequence processing will be extensively presented in sec. 7, while ARIMA will be presented in the Method section. Before going through neural sequence processing, it is necessary to start with an introduction to the basics of machine learning and training of neural networks in sections 5, 6.

5 Introduction to Machine Learning

Machine learning (ML) is the discipline that studies how to optimize the parameters of a model, based on a collection of data and a performance measure that quantifies how well the model is processing the data at hand (Goodfellow, Bengio, & Courville, 2016). Machine learning is typically employed on tasks that are too complex to feasibly solve using a fixed model. The difference between the two approaches is that a fixed model will explicitly program how to do a task, while an ML model will *learn* how to do the task based on the input it is given. Machine learning problems are generally categorized as *supervised* or *unsupervised*, referring to if the model uses labels (true values) or not, to measure its performance and update the model. When having access to true values, the model learns how to reproduce the correct output given a specific input. The model then uses its performance, the error measured between the estimated and correct labels, to improve the model. Unsupervised learning is typically applied to types of problems and data that are either unpractical or infeasible to label. Such problems often do not have the objective of finding the most correct mapping, but to learn hidden groupings and patterns seen in the data.

A neural network is a machine learning architecture modeled and inspired by the structure of the brain, where multiple nodes or *neurons* are interconnected within and across network layers. Deep learning involves the use of stacked neural networks to learn complex tasks and is categorized as a field within machine learning (Choi, Coyner, Kalpathy-Cramer, Chiang, & Campbell, 2020). The accelerated growth of machine learning, and more specifically deep learning, are motivated by the increased capability to collect and store extensive amounts of data while also seeing the rise of increased processing power (Jordan & Mitchell, 2015). Large amounts of data are essential for the performance of deep learning models as the capability to learn increases with the size of the training data.

6 Training a Neural Network

Neural networks are composed of different computational *layers* which it is common practice to categorize as input layer, hidden layers, and output layer. The depth of a neural network is determined by the total number of layers, and a *deep neural network* is, therefore, one consisting of several layers. A deep neural network is not necessarily the solution to every problem and the depth is determined by the problem at hand.

Network layers can take many forms depending on the model. In general, a network layer takes the output from the previous layer as input and maps it through a combination of *learnable parameters* passed through a non-linear *activation function* and sends this mapping to the next layer. The process of running a neural network from the input layer to the output layer is commonly called *forwarding*.

Training a neural network involves minimizing the *loss function*, which for supervised learning computes the error between the predicted and actual values. This error is propagated backwards in the network by computing the derivative of the loss function with

respect to each parameter in a process called *backpropagation*. The derivative is often called the *error gradient*, or simply just the gradient.

The gradient is used to update the learnable parameters using an *optimization algorithm* which adjusts the parameters such that the loss function is minimized. The process of minimizing the loss function is called *gradient descent*. This iterative learning process continues until a minimum is reached. Each cycle of the forward and backward passes is called an *epoch* or run.

The following sections will delve further into the key concepts necessary to comprehend the potential challenges associated with training neural networks.

6.1 Network Optimization

Network optimization is a rich field of research that is devoted to finding solutions to make network convergence faster. This is done by investigating the loss surface of a network and how the surface is traversed given different network optimization algorithms. This is important because a loss surface can be complex and although only one point can be the *global minimum* there usually exist several *local minima* where the gradient can get stuck. An extensive study by (H. Li, Xu, Taylor, & Goldstein, 2017) visualized and investigated the effect different architectures have on the loss surface. They found that loss surfaces get more complex and non-convex the deeper the model. This emphasizes the need for good optimization practices when working with very deep models.

6.1.1 Initialization

The optimization process starts with random initialization of the network's learnable parameters. These parameters are called *weights* and *biases*. Proper initialization has shown to be essential for the quick convergence of neural networks and several different initialization techniques have been developed, and which one to choose depends on the kind of activation functions present in the network (Glorot & Bengio, 2010). Common to most initialization techniques is to initialize to small values chosen from some form of probability distribution. In (H. Li et al., 2017) they argued that the reason why this has an impact on convergence time is due to where the parameters are initialized on the loss surface. If initialized in highly complex terrain and not close to a convex area, gradient descent updating will have poor results in finding a strictly decreasing area resulting in slow convergence.

6.1.2 Network Optimization Algorithms

Network optimization aims to minimize the loss function with the desire of improving the performance measure. Even though the goal is to minimize the loss function, it is not necessarily advantageous, or possible, to find the global minimum, and a local minimum can often serve the task sufficiently (Chilimbi, Suzue, Apacible, & Kalyanaraman, 2014). Nevertheless, finding the global, or a local, minimum requires optimization algorithms which update the model parameters using the derivative of the loss function in a process called gradient descent.

Basic gradient descent updates the weights and biases as follows (Goodfellow et al., 2016):

$$\theta_{\mathbf{i+1}} = \theta_{\mathbf{i}} - \eta \nabla_{\theta_{\mathbf{i}}} J(\theta_{\mathbf{i}}) \tag{8}$$

where θ represents the learnable parameters, $J(\cdot)$ represents the loss function and η represents the learning rate, a hyperparameter determining the step of which the parameters are updated.

Due to the fixed learning rate used in basic gradient descent, the updates to the weights were often too small or too large resulting in either too slow convergence or making convergence infeasible as the weights often oscillated around the minimal point. Improvements to the basic gradient descent have come in the form of momentum factors giving the opportunity to use small learning rates with the addition of a momentum factor which further pushes the learning in the same direction as previous updates. Essentially, it creates an adaptive updating step where the algorithm assumes that the previous updates were in the right direction.

The historically most popular optimization algorithm, stochastic gradient descent (SGD), improves on the basic gradient descent by randomly choosing subsets of the training dataset to compute the gradient on. The reason for using this stochastic method is to reduce the computational cost of training when the training dataset grows large. Another way to achieve adaptive updating of the gradient is in the form of adaptive learning rates which is the basis of popular optimization algorithms such as ADAM and ADAGRAD. Adaptive learning rate involves reducing the size of the learning rate during optimization such that the learning rate is sufficiently small when nearing the minimal point with the goal of reducing oscillations.

6.1.3 Residual Connections

As stated at the beginning of this section, very deep neural networks showed to be difficult to optimize resulting in the development of *residual connections* (He, Zhang, Ren, & Sun, 2016) also called *skip connections*. A residual connection can be thought of as a shortcut, connecting the input of one layer to the output of a subsequent layer. Such layers eased the training by making the intermediate layers learn the residual mapping, in other words, the difference between input and output, instead of the full mapping. In (H. Li et al., 2017) they compared the loss surfaces of neural nets with and without residual connections, showing that residual connections resulted in more convex loss surfaces making optimization easier. Therefore neural networks can be made deeper without falling into the caveats of complex loss surfaces.

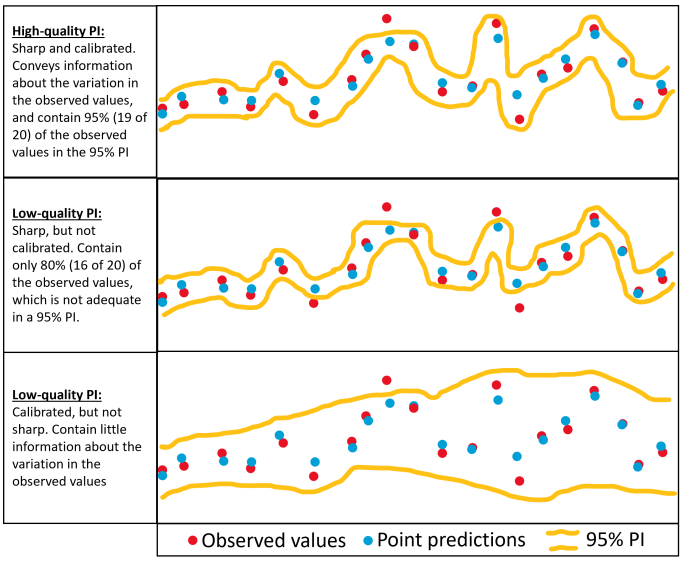


Figure 2: Overfitting and underfitting in probabilistic time series forecasting (Eikeland et al., 2022)

6.2 Network Regularization

An important aspect to consider when training neural networks is their ability to *generalize well on unseen data*. When the model fails to generalize, the effect is popularly called *overfitting* and is a situation that arises when the model performs very well on the subset of data on which it was trained, while failing to perform well on new data. The job of training neural networks that generalize well is called *network regularization* and, as counterintuitive as it sounds, relies on simplifying the models such that it is unable to overfit the training data.

In the probabilistic time series forecasting problem, overfitting takes the form of too-sharp prediction intervals. While the opposite, *underfitting*, takes the form of too wide prediction intervals. Both examples are depicted in Fig. 2 where overfitting is the middle panel, while underfitting is the bottom panel. The top panel shows a proper 95% PI.

To monitor a model's ability to generalize it is typical to split the data into three parts called *training, validation* and *testing* datasets. The training dataset is used to train the model while the validation dataset is used to validate the model during training. When the model is finished training the model is tested against the testing dataset. The error calculated against the testing dataset is often called the *generalization error* and this is the error metric reported when presenting the performance of neural networks.

6.2.1 Early Stopping

The most straightforward regularization technique relies on stopping the training before the model learns to overfit. This is done by monitoring the model performance on the validation dataset during training such that training can be stopped when the performance on the validation dataset starts deteriorating. It is typical to plot the loss on the training dataset against the loss on the validation dataset to identify deviating results on the validation dataset. From the plot, one should be able to identify at which epoch the training should be stopped. In fig. 3 it is clear that the model starts overfitting to the training dataset around epoch 100 and it is therefore of no use training beyond this point.

6.2.2 Dropout

Dropout is a regularization technique that relies on dropping random parts of the network during training. This method was developed originally for fully connected layers, where all nodes in one layer are connected to all nodes in the succeeding layer. Dropout then relies on randomly dropping nodes with a probability of p. The rationale behind this is to prevent the nodes from being overly dependent on the output from previous layers and to rather learn from irregular information. This makes the network more robust against new information.

Dropout showed to be a simple and effective regularization technique and has been adapted to different neural network models. Although the method cannot be directly adapted, all the variations depend on simplifying the network in some way during training.

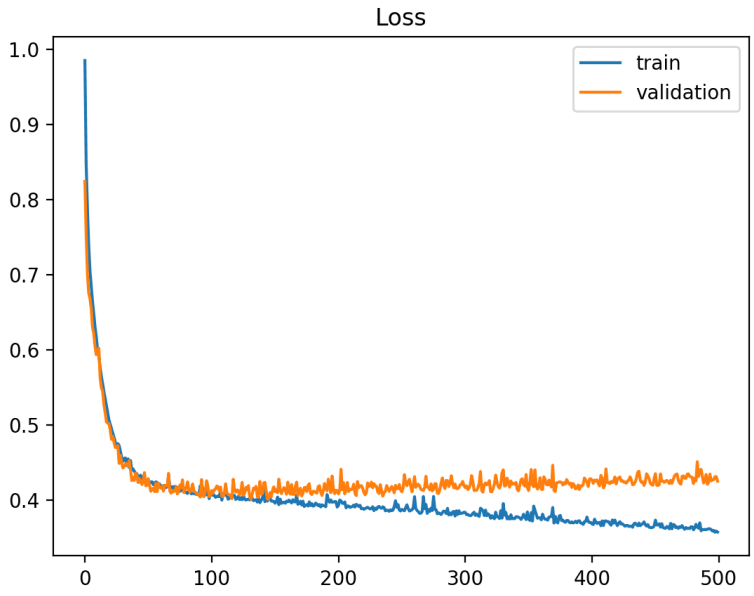


Figure 3: Validation and training loss of an overfit model (Brownlee, 2019)

6.2.3 Weight decay

Another way to regularize the network is by modifying the loss function by adding penalties to the weights. There are two popular penalties, L1 and L2 regularization, where L2 regularization is commonly known as *weight decay* (Goodfellow et al., 2016).

The general function for adding weight penalties is to modify the loss function as follows:

$$J'(\theta) = J(\theta) + a\Omega(\theta) \tag{9}$$

Where $\Omega(\cdot)$ is the norm penalty function and a is a constant determining the effect of the penalty on the loss function. The constant a is typically lower than the learning rate.

The most popular norm penalty is weight decay where the penalty takes the form of $\frac{a}{2}||\theta||^2$. The gradient then becomes (Goodfellow et al., 2016):

$$\nabla J'(\theta) = \nabla J(\theta) + a\theta \tag{10}$$

And the updating step becomes:

$$\begin{aligned} \theta_{\mathbf{i+1}} &= \theta_{\mathbf{i}} - \eta\nabla_{\theta_{\mathbf{i}}} J'(\theta_{\mathbf{i}}) \\ \theta_{\mathbf{i+1}} &= (1 - \eta a)\theta_i - \eta\nabla_{\theta_i} J(\theta_i) \end{aligned} \tag{11}$$

Adding weight decay restricts the updating step by restraining the weight from becoming too large and thus reduces the impact of large weight values. Similarly as with dropout, weight decay inhibits the ML model from overfitting by forcing the model to learn more robust representations.

7 Neural sequence processing

While time series prediction has had an important role in many industries, neural sequence modeling has become increasingly developed due to natural language processing (NLP). Most neural sequence models are made for tasks such as machine translation, text generation, sentiment analysis, and speech recognition. This section will cover the basics of two popular neural sequence processing models: the recurrent neural network (RNN) and the Transformer. While RNNs are a mature family of models, the Transformer was first introduced in 2017 by (Vaswani et al., 2017). The two models have quite different approaches to the problem, where RNNs process the data sequentially and the Transformer process the data as a set.

7.1 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are specialized at processing sequential data and have historically been a basic building block for models dealing with sequence tasks (Bianchi, Livi, & Alippi, 2016). As the name suggests, RNNs are recurrent in nature meaning that the same process is performed for every input to the network. Given the terminology established in sec. 4, a recurrent unit calculates the following operation for each target-covariate pair y_t, x_t (Lim & Zohren, 2021):

$$\begin{aligned} h_t &= f_h(\mathbf{W}_1 h_{t-1} + \mathbf{W}_2 y_t + \mathbf{W}_3 x_t + \mathbf{b}_1) \\ \hat{y}_t &= f_y(\mathbf{W}_4 h_t + \mathbf{b}_2) \end{aligned} \tag{12}$$

Where $\mathbf{W}_{(\cdot)}$ represents weights, $\mathbf{b}_{(\cdot)}$ represents biases and $f_{(\cdot)}$ represents activation functions. This recurrent process utilizes memory in the form of the hidden state h_t, which is updated by taking into account the current input and previous hidden state. The RNN must learn what temporal information to store and what to omit and this is what the hidden state is designed to do when taking both the current input and the previous hidden state as input to the recurrent unit.
As eq. 12 indicate, an RNN has the same amount of recurrent units as the input sequence has elements and parameters are shared across units (Bianchi, Maiorino, Kampffmeyer, Rizzi, & Jenssen, 2017). This parameter sharing makes it possible to extend to sequences of different lengths and RNN layers have therefore no need for the look-back window introduced in section 4.

Deep RNNs are made by stacking several layers such that a recurrent unit feeds the output $\hat{y}_t$ as input to the corresponding time step in the next layer and the hidden state is passed to the next time step in the current layer.

Although the recurrent operation was introduced as the solution to capture temporal information in deep learning models, the original *vanilla RNN* showed to be bad at capturing long-term dependencies. The recurrent unit requires cascading calculations of the backflowing gradient resulting in a problem known as the *vanishing/exploding gradient* (Hochreiter, 1998) which will be presented in the next section.

7.1.1 Vanishing and Exploding Gradients

The vanishing/exploding gradient is a weakness of the *backpropogation through time* (BPTT) algorithm, which is an extension of backpropogation specifically designed for training RNNs. While this problem is not specific to RNNs, it becomes a frequent issue for these models when the number of computations required to learn long-term dependencies becomes excessively large (Hochreiter, 1998).

Repeating gradient updates with either too small or too large weight updates tends to respectively result in the gradient vanishing or exploding. Due to weight sharing, the weight will be multiplied by itself for each element in the sequence. If the weight then is small, this will result in the value exponentially decaying to zero. Similarly, if the weight is too large the value will exponentially grow to infinity. Both are problematic and will effectively reduce the ability of the network to optimize and learn dependencies (Goodfellow et al., 2016).

An easy solution for exploding gradients is *gradient clipping*, also known as *norm clipping*. This method involves scaling down the gradients when the gradient reaches a predetermined threshold (Pascanu, Mikolov, & Bengio, 2013). This is a simple and efficient solution to exploding gradients. Other solutions can be regularization techniques such as L1/L2 regularization, or using randomized architectures that do not train the recurrent part (Bianchi, Scardapane, Løkse, & Jenssen, 2020; Bianchi, Scardapane, Uncini, Rizzi, & Sadeghian, 2015; Bianchi, De Santis, Rizzi, & Sadeghian, 2015).

Dealing with the vanishing gradient is more complicated and does not have as straightforward solutions as with exploding gradients. (Hochreiter & Schmidhuber, 1997) developed the long short-term memory (LSTM) network to address the vanishing gradient by creating more advanced recurrent units where the gradient could flow more freely. Although LSTM is the most popular, a more recent option termed gated recurrent unit (GRU) (Cho, van Merrienboer, Bahdanau, & Bengio, 2014) is also available. The two choices differ in the number of parameters yet are not dissimilar. An empirical exploration between the two types can be found in (Jozefowicz, Zaremba, & Sutskever, 2015). When talking about an RNNs recurrent unit, one is almost always referring to a gated unit, either the LSTM or the GRU.

7.1.2 Long Short Term Memory (LSTM)

The gradient flow in LSTM units is managed through the introduction of gates giving the opportunity for the long-term memory to be regulated. The gates in an LSTM unit are called the *input gate*, the *output gate*, and the *forget gate*. The recurrent operations in an LSTM are calculated with the following 3 equations (Lim & Zohren, 2021)

$$\begin{aligned}
\mathbf{i_t} &= \sigma(\mathbf{W_{i,1}h_{t-1}} + \mathbf{W_{i,2}}y_t + \mathbf{W_{i,3}x_t} + \mathbf{b_i}) \\
\mathbf{o_t} &= \sigma(\mathbf{W_{o,1}h_{t-1}} + \mathbf{W_{o,2}}y_t + \mathbf{W_{o,3}x_t} + \mathbf{b_o}) \\
\mathbf{f_t} &= \sigma(\mathbf{W_{f,1}h_{t-1}} + \mathbf{W_{f,2}}y_t + \mathbf{W_{f,3}x_t} + \mathbf{b_f}),
\end{aligned} \tag{13}$$

where $\sigma(\cdot)$ is the sigmoid activation function mapping the output to the interval $[0, 1]$. A blocked gate will take the value 0, while a completely open gate takes the value 1. In practice, the gates rarely take these extreme values but lie somewhere in between (Greff, Srivastava, Koutník, Steunebrink, & Schmidhuber, 2016).
Information is passed forward through the hidden state and the cell state:

$$\begin{aligned} \mathbf{c_t} &= \mathbf{f_t} \odot \mathbf{c_{t-1}} + \mathbf{i_t} \odot tanh(\mathbf{W_{c,1}}\mathbf{h_{t-1}} + \mathbf{W_{c,2}}\mathbf{y_t} + \mathbf{W_{c,3}}\mathbf{x_t} + \mathbf{b_c}) \\ \mathbf{h_t} &= \mathbf{o_t} \odot tanh(\mathbf{c_t}) \end{aligned} \tag{14}$$

where $\odot$ is the element-wise product and $tanh(\cdot)$ is the tanh activation function.

Even though RNNs are improved by the inclusion of gated units, there still remain some issues concerning the recurrent nature of the networks reported by (Bai, Kolter, & Koltun, 2018). (i) RNNs cannot compute in parallel due to having to wait for the preceding units making them slower to train than, e.g., convolutional neural networks (CNNs) or Transformer networks, (ii) LSTM or GRU units consume much of the computer's memory when processing long sequences and (iii) gated units still struggle with long term memory and give sub-optimal results for sequence lengths longer than 50-200 for LSTM and GRU. The reader is directed to (Bai et al., 2018) for further information.

7.1.3 Sequence to Sequence structure

An important application of RNNs is in sequence-to-sequence (seq2seq) models (Sutskever, Vinyals, & Le, 2014), which originally were designed for machine translation. As the name suggests, a seq2seq model maps a sequence to another sequence. The model consists of one *encoder* and one *decoder* connected by a *context vector*. Although the encoder and decoder can be any neural sequence model, they have historically been deep RNNs (Sutskever et al., 2014; Cho et al., 2014; Kalchbrenner & Blunsom, 2013). The motivation behind using encoders and decoders is to train them on different tasks. In the machine translation problem, the encoder is trained on encoding a word or sentence in one language. The encoded information is retained in the context vector of fixed size which is passed to the

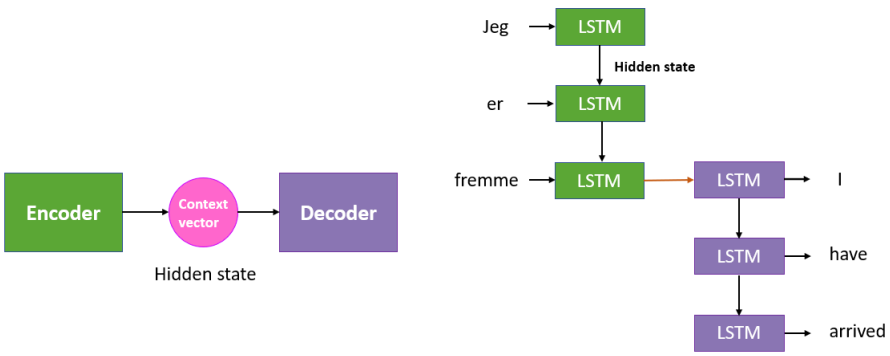


Figure 4: Simple illustration of the encoder-decoder structure

decoder who is trained on decoding the information in the vector (Cho et al., 2014). Fig. 4 shows a basic illustration of the structure of a seq2seq model.

For time series forecasting, the seq2seq structure can be utilized by having the encoder take inputs from the look-back window, τ, and the decoder take the known future covariates in addition to the hidden state and output from the previous layer. In (Salinas, Flunkert, Gasthaus, & Januschowski, 2020) they make a point of using the same RNN in both the encoder and decoder due to being trained on similar data as opposed to translating from one type of data to another.

A disadvantage of the seq2seq model relates to the fixed size of the intermediate context vector, which makes it act as a bottleneck. A too-small context vector will not be able to retain the information in a long sequence (Goodfellow et al., 2016), and a too-large context vector will have to pad the shorter sequences.

7.2 Transformer models & The Attention Mechanism

At the heart of Transformer models is the attention mechanism, and even though most of the Transformers to date have dissimilar architecture, they are all in some way based on attention. The attention mechanism is an order-agnostic way to learn temporal dependencies (Tsai, Bai, Yamada, Morency, & Salakhutdinov, 2019), as opposed to the recurrent operations of RNNs. The name Attention comes from the ability to focus (the attention) on relevant parts of the input sequence (Vaswani et al., 2017).

7.2.1 Scaled dot product Attention

The most common approach to attention is 'scaled dot product Attention', its computational graph is depicted in Fig. 5a. Common to all Attention calculations is the mapping of three vectors, Query, Key, and Value, to an output vector. One can think of the **Query** vector as a question vector, answered by the **Key** vector. From this compatibility, a Softmax value is calculated and multiplied with the **Value** vector which is designed to carry information forward. Scaled dot product Attention is calculated as follows:

$$\text{Attention}(\mathbf{x_q}, \mathbf{x_k}, \mathbf{x_v}) = \text{Softmax}\left(\frac{\mathbf{x_q} \cdot \mathbf{x_k}^T}{\sqrt{d_k}}\right)\mathbf{x_v}, \tag{15}$$

where $\mathbf{x_q} \in \mathbb{R}^{d_k}$, $\mathbf{x_k} \in \mathbb{R}^{d_k}$, $\mathbf{x_v} \in \mathbb{R}^{d_v}$, and d_k, d_v is the feature dimensions of $\mathbf{x_k}, \mathbf{x_v}$. In practice, Attention is calculated simultaneously where the query,key and value vectors are features in matrices, $\mathbf{X_q}, \mathbf{X_k}, \mathbf{X_v}$.

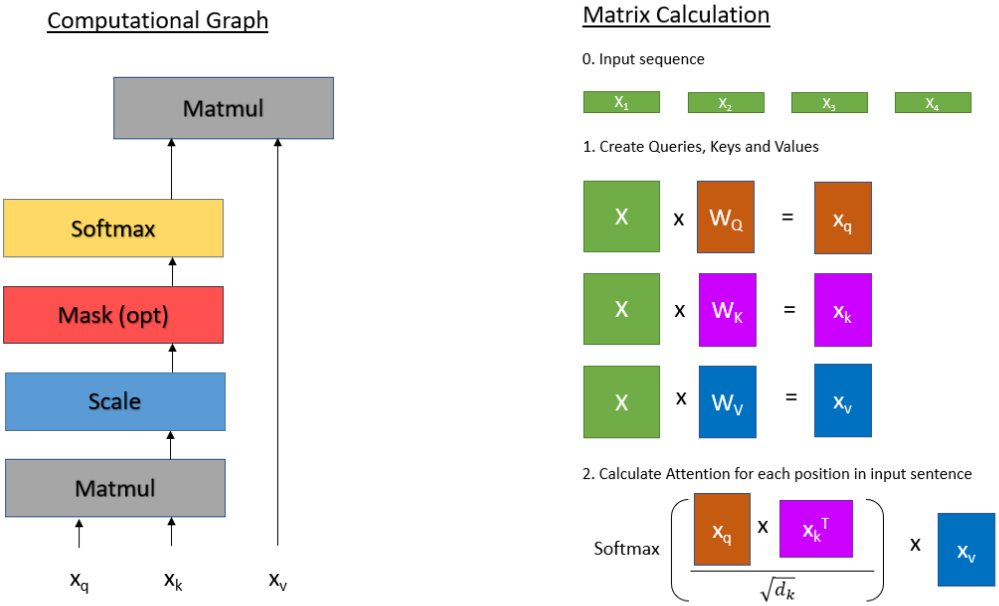
7.2.2 The Queries, Keys, and Values

The Queries, Keys, and Values matrices are created by first stacking the input features to one input matrix, $\mathbf{X} \in \mathcal{X}$, where $\mathcal{X}$ denotes the feature space. The input matrix is linearly transformed using weight matrices, $\mathbf{W_Q^{(0)}}$, $\mathbf{W_K^{(0)}}$, $\mathbf{W_V^{(0)}}$ to create the query, key and value vectors

$$\mathbf{X_q} = \mathbf{X}\mathbf{W_Q^{(0)}}, \quad \mathbf{X_k} = \mathbf{X}\mathbf{W_k^{(0)}}, \quad \mathbf{X_v} = \mathbf{X}\mathbf{W_v^{(0)}}, \tag{16}$$

as illustrated in step 1 in Fig. 5b. The Query X_q, Key X_k, and Value X_v matrices are created such that the first row of the matrices corresponds to the first item in the input sequence and so on. By taking the matrix multiplication of the Queries and Keys, each query vector will be checked against all the key vectors such that one row in the intermediate output matrix, $\mathbf{X_q}\mathbf{X_k}^T$, correspond to the same row in the input matrix. This means that all the elements in the input are checked against every position. Intuitively, the product is then a measure of how compatible each element is with each position. Taking the Softmax gives a score of which elements are most related to the position.

The final output of the Scaled dot product Attention is then a matrix with as many rows as the input matrix. The first row corresponds to the first item of the input sequence, and the second row to the second item, etc. Each row is a product of the value vectors multiplied by the Softmax values.



(a) Computational graph of scaled dot product attention

(b) Illustration of query, key and value matrices and the scaled dot product attention

Figure 5: Calculations behind the Attention mechanism

7.2.3 Multi-Head Attention

Attention is typically implemented within an operation called Multi-head attention. This gives the opportunity to apply several Attention mechanisms at once, which improves the performance similarly as ensemble methods improve the performance by collecting several point forecasts. Additionally, applying several attention mechanisms increases the representation subspaces, giving the possibility to focus on different parts of the input sequence (Vaswani et al., 2017).

The Queries, Keys, and Values are calculated as discussed previously, however, they are subsequently linearly transformed into h different parallel layers or 'heads'. This results in h different Query-, Key- and Value-matrices which gives h Scaled dot product Attention calculations. The formula is

$$\begin{aligned} \text{Multihead}(\mathbf{X_q}, \mathbf{X_k}, \mathbf{X_v}) &= \text{Concat}(\mathbf{H_1}, \mathbf{H_2}, \ldots, \mathbf{H_h})\mathbf{W_0} \\ \mathbf{H_i} &= \text{Attention}(\mathbf{X_q}\mathbf{W_Q^{(i)}}, \mathbf{X_k}\mathbf{W_K^{(i)}}, \mathbf{X_v}\mathbf{W_V^{(i)}}), \end{aligned} \tag{17}$$

where $\mathbf{W_Q^{(i)}} \in \mathbb{R}^{d_{model} \times d_k}$, $\mathbf{W_K^{(i)}} \in \mathbb{R}^{d_{model} \times d_k}$, $\mathbf{W_V^{(i)}} \in \mathbb{R}^{d_{model} \times d_v}$, $\mathbf{W_0} \in \mathbb{R}^{hd_v \times d_{model}}$.

The output vectors from each Attention calculation are concatenated to the model dimension, d_{model}, which in the original transformer was 512. Due to the concatenation and the requirement that each layer has the same output dimension, each Attention head will have reduced dimensions compared to having one single Scaled dot product Attention. Therefore the dimensions are as follows:

$$d_{model} = d_v \times h \implies d_v = d_{model}/h.$$

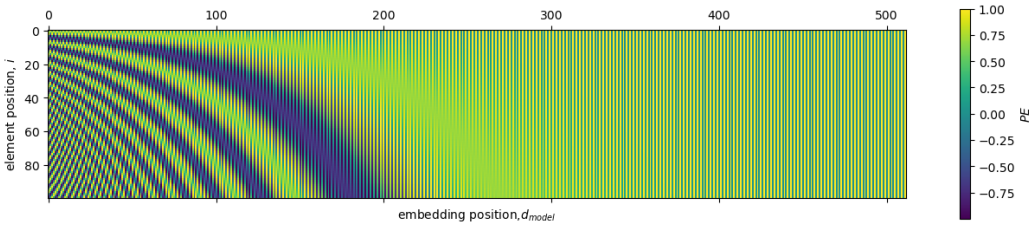


Figure 6: Positional encoding matrix with n=10000 and $d_{model} = 512$ as in the original Transformer by (Vaswani et al., 2017)

7.2.4 Positional encoding

The attention mechanism is an order-agnostic operation, meaning that the sequence is treated as a set and the order of the input does not change the output. It is therefore necessary to provide additional positional encoding to the input. The most common way to represent position is as a combination of sine and cosine functions

$$\begin{aligned} PE_{\text{pos},2i} &= \sin\left(\frac{\text{pos}}{n^{2i/d_{model}}}\right) \\ PE_{\text{pos},2i+1} &= \cos\left(\frac{\text{pos}}{n^{2i/d_{model}}}\right), \end{aligned} \tag{18}$$

where pos denotes the position of the element in the input sequence, i denotes the position along the model dimension d_{model}, and n is a user-defined scalar. The positional encoding matrix is described in eq. 18 and visualized in Fig. 6. There are some benefits of using the positional encoding of sine and cosine as opposed to e.g. integer positional value. With input sequences of varying lengths, an integer value would give different normalization while the sine/cosine functions are bounded by $[-1, 1]$.

8 Wind Power Forecasting for the Electricity Market

It is necessary to know some basic background on wind power and the electricity market to both make the best possible wind power forecasts and to get the most out of the forecasts. Investigating wind power in general gives insight into which parameters are important for the power output, and having knowledge of the electricity market makes the operational framework clearer.

8.1 Wind Energy Production

The kinetic power [W] that the wind carries is

$$P_k = \frac{1}{2}\dot{m}v^2 = \frac{1}{2}\rho A v^3, \tag{19}$$

where $\dot{m}$ is the mass flow rate of air, which can be derived by the density of the air, ρ, the swept area of the turbine blades, A, and the velocity of the wind hitting the turbine, v. Further, air density is defined by the ideal gas law and is dependent on air pressure, temperature, and the ideal gas constant of the air. Therefore, meteorological variables relevant to wind power production are wind velocity, air pressure, and temperature. Most dominating is the wind velocity due to the cubic relationship.

Wind can be decomposed into three velocity components: zonal, meridional, and vertical. The positive and negative zonal direction respectively corresponds to westerly and easterly winds and the positive and negative meridional direction respectively corresponds to southerly and northerly winds (Wallace & Hobbs, 2006). It is important to differentiate between the zonal and meridional wind compositions when considering wind power, seeing as the direction of the wind is essential for how much wind actually hits the turbines.

Physical limitations to wind energy production are turbine-specific variables such as turbine sweep area, A, and the capacity coefficient of the turbine. Therefore the power extracted by a wind turbine can be expressed as

$$P_t = C_p P_k, \tag{20}$$

where C_p represents the power coefficient of the turbine. The power coefficient is upper bounded by the theoretical Betz limit of 16/27 or approximately 0.59% (Manwell, McGowan, & Rogers, 2010).

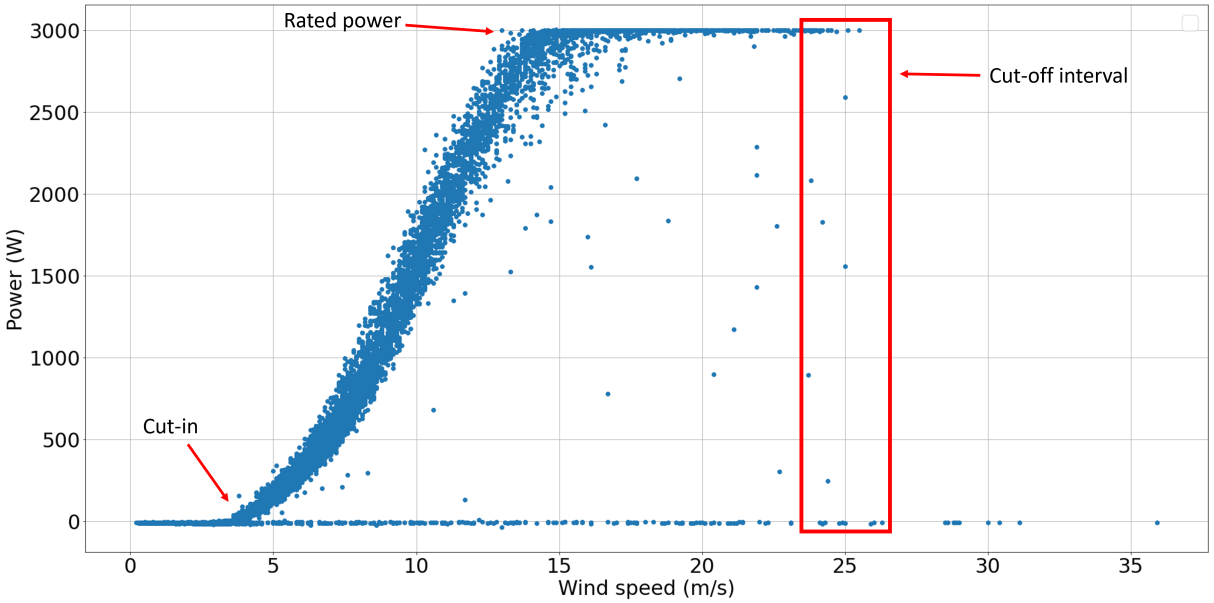


Figure 7: In-situ power curve from a wind turbine in Fakken wind farm with cut-in, rated, and cut-out wind speed specified (Eikeland et al., 2022).

8.1.1 Power Curve

The power curve is a description of how much power a specific wind turbine produces given different wind speeds. Important aspects of the power curve are the *cut-in*, *rated* and *cut-out* wind speeds which respectively are the wind speeds where the turbine starts producing power, reaches maximum rated power, and stops producing power. The manufacturer delivers a power curve for each wind turbine type, with values derived from stable and ideal conditions which most likely do not reflect the conditions on site. Fig. 7 shows the actual measured power curve from a wind turbine in the Fakken wind farm giving a more nuanced view of the conditions. The figure depicts the cut-in wind speed of around 4 m/s, followed by the non-linear relation before hitting the rated wind speed of around 12-13 m/s and thereafter the cut-off interval of around 25 m/s. Also visible are points where the turbine has been out of production due to turbine failure or maintenance.

8.2 The Nordic energy Market

Electricity is a special commodity compared with other commodities. (Wangensteen, 2012) lists the following features of electricity: (*i*) *continuously flowing*, i.e. continuously consumed and produced, (*ii*) *instantly generated and consumed*, (*iii*) *limited storage options*, (*iv*) *consumption variability*, the consumption pattern varies with time of day, day of week and week of the year, (*v*) *non-traceability*, meaning there is no physical way to distinguish which energy unit came from which generator, (*vi*) *community essential* and (*vii*) *breakdown possibility*, meaning that if not handled correctly, the grid can break down.

Due to the above-mentioned features of electricity, a common ground is needed for efficient and safe trading. Today most of the electricity exchange of Europe runs through Nord Pool spot, which originated as a Norwegian electricity exchange before being the first

multinational electricity exchange when Sweden joined in January 1996 (Wangensteen, 2012). Nord Pool acts as the Market Operator (MO) and is a completely open and deregulated market for selling and buying electricity. A deregulated market means that the market participants compete freely without involvement from the state, and hence the prices are determined by supply and demand.

The MO is responsible for matching bids and sales. The process of clearing prices depends on the length of the trading time frames. The different time frames are depicted in Fig. 8. Physical electricity trading takes place in the ahead markets, Elspot and Elbas, and the balancing market is managed by the Transmission System Operator (TSO). Financial trading is handled by Nasdaq and involves contracts named futures and forwards which generally have a longer trading frame than Elspot and Elbas. Although the prices are settled earlier, the actual physical trading still takes place in the day-ahead market and the customer pays the settled price instead of the spot price.

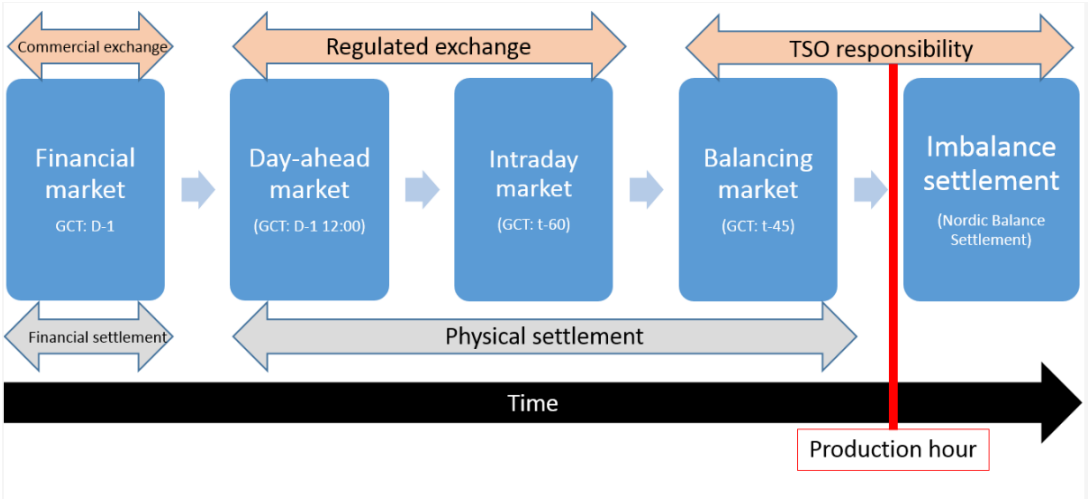


Figure 8: Trading time frames in the Norwegian electricity market (NVE, 2021)

8.2.1 Elspot: day-ahead trading

Day-ahead physical trading is managed through the Elspot market, which uses periodic clearing to clear prices. Periodic clearing involves settling the prices in one process instead of continuously. Implying that, in the day-ahead framework, prices are settled for all hours on the following day.

At 10:00 CET, Nord Pool announces the available capacities in the grid, and by 12:00 CET sellers and buyers must have submitted their hourly bids (NordPool, n.d.). Bids for purchasing electricity must specify the maximum price the customer is willing to pay given the hour and area, while bids for selling electricity must specify the minimum price the producer is willing to sell for given the hour and the area. Nord Pool clears bids by finding the intersecting point between aggregated sale and purchase curves for one hour at a time. In an optimal market, all areas would pay the market clearing price which is the price found in an unconstrained system. However, transmission limitations resulting in congestion between areas results in different clearing prices called area prices.

8.2.2 Elbas: intra-day trading

Intra-day trading is managed through the Elbas market, which operates as the aftermath of day-ahead trading (Wangensteen, 2012). The Elbas market enables trading up to one hour before delivery giving the opportunity to adjust for inaccurate bids given during the day-ahead trading. While sometimes referred to as a balancing market, the Elbas market must not be confused with the balancing markets operated by TSOs. Elbas is an intra-day market for market participants like consumers and producers, while the TSO balancing market deals with imbalances in the grid.

8.2.3 The Market for Reserves Acquisition

As already mentioned, the balancing market is operated by the TSO of each country. The purpose of the balancing market is to have reserves ready for events resulting in frequency and voltage imbalances. Such events can be an imbalance between supply and demand, outages of transmission lines or system faults. To acquire reserves, the TSO purchases generation from producers prior to the day-ahead market clearing. In Norway, Statnett is the TSO and they secure the available capacities a week ahead of the operation. The Norwegian reserves are divided into primary, secondary and tertiary reserves. The primary reserves are automatic and respond during a couple of seconds of a frequency or voltage imbalance, while the secondary and tertiary reserves are manual. They have a respective response time of a couple of minutes and 10-30 minutes and are set in place to take over the load of the primary reserves so they are available to respond to new imbalances (Wangensteen, 2012).

8.3 Introduction to Wind Power Forecasting

Wind power forecasting (WPF) models can be classified either by their forecasting horizon, i.e. short-term, medium-term, or long-term, or by their applied methodology. Classifying WPF models by forecasting horizon essentially classifies them by application purposes. (Hanifi, Liu, Lin, & Lotfian, 2020) lists the following time horizons and corresponding applications: (*i*) very short-term, $\sim$ 30 min, applies to the real-time grid and regulatory operations, (*ii*) short-term, 30 min - 6 h, applies to load dispatch planning and load intelligent decisions, (*iii*) medium-term, 6 h - 24 h, applies to energy trading and operational security, and lastly (*iv*) long term, 1 day to a month, applies to maintenance schedules, optimum operating cost and operational management.

Forecasting wind power for the day-ahead market will involve a forecasting horizon of 12-36 h and is therefore classified as either medium or long-term forecasting. Forecasting for the intraday market will be classified as either short-term or medium-term as forecasting horizons can range from 1-24 h. However a smaller horizon from 1-12 h is often most advantageous as the intraday market acts as a "correcting" market for the producers and consumers and smaller horizons generally lead to higher quality forecasts.

Classifying WPF models by methodology leads to distinguishing models either as physical or statistical methods (Hanifi et al., 2020). Physical methods involve calculating wind power using the turbines' power curve and wind speed derived from complex mathematical models created using surrounding topography. Statistical methods can be further divided into classical time series models and artificial neural networks (ANNs), which are the kind of models this study will focus on.

Part III / Proposed Method

The aim of this thesis is to answer the research questions formulated in Section 2:

- Is it possible to provide accurate wind power forecasts that reflects the uncertainty of the predictions at Fakken wind park using two wind power forecasting models trained to forecast respectively to the day-ahead and intraday market?

- Is the forecasting library Pytorch-Forecasting (Beitner, 2020) ready for commercial use by users that are not experts in machine learning?

A case study based on real data is investigated using wind power forecasting (WPF) models with a focus on deep learning. The output of the models will be used as input to help guide the wind power producer to reduce uncertainty and maximize their potential bidding to the energy market. Sec. 9 will present the operational framework of the WPF models, while the models themselves will be presented in sec. 11.

9 The Operational Framework

9.1 The day-ahead market framework

Sec. 8.2.1 introduced the day-ahead market and its bidding deadline is specified to be at 12:00 CET which will also be used in the day-ahead market framework of this thesis. This entails that the physical time of forecasting, hereby called prediction time, for the day-ahead market happens 12-36 hours before production time. The consequences of this are twofold:

1. The forecasting problem becomes a *cold start* forecasting problem, meaning that forecasts are made solely from known covariates as historical production from 12:00-24:00 the day before has not happened yet at prediction time. Forecasting with a longer forecasting horizon, i.e. from 12:00 at prediction time to 24:00 the day after would not have made use of the 'available' historical production data from before prediction time due to the low memory seen in wind power data.

2. The available numerical weather predictions are made at an early stage resulting in increased uncertainty in the input to the forecasting model.

Due to the limitations associated with day-ahead forecasting, a high uncertainty forecast is expected. To accommodate for the high uncertainty, the wind power producer is advised to bid from a low quantile of the probabilistic forecast to minimize the risk of overbidding. The intraday market will be used to 'correct' the bids made during the day-ahead trading.

9.2 The intraday market framework

Sec. 8.2.2 introduced the intraday market where bidding is possible up to one hour before production time. Forecasting for the intraday market will therefore have the possibility to use the look-back window with historical power production. Since predictions can be made at any time, this opens up the possibility of a smaller forecasting horizon which generally is associated with lower uncertainty. This study will test forecasting for the intraday market with 6-hour forecast horizon, meaning that four forecasts are made during one day.

9.3 Data

This thesis examines historical wind power data and numerical weather prediction (NWP) data as the input variables to deep learning models predicting future wind power production. The available data is presented in the next sections.

9.3.1 Wind power data

The wind power data comes from Fakken wind park which is located on the island Vannøya in Troms, Norway. Its total capacity is 54 MW, distributed on 18 wind turbines each with a maximum capacity of 3 MW and 80 m hub height (TromsKraft, n.d.). The park is located in complex terrain, and the owner of the wind power plant reported that there was a 25% difference in production from two different turbines due to the topology at the farm (Eikeland et al., 2022). Additionally, the surrounding islands are also characterized by mountainous terrain having an impact on the wind resources at Fakken.

Historical power output data consist of the total power output for the whole park for the years 2017-2020. The power data is the same used in (Svane, 2022), and is publicly available data retrieved from NVEs websites.

9.3.2 Meteorological weather data

An additional dataset of numerical weather prediction (NWP) is constructed using data provided by the Meteorological Institute of Norway (MET Norway) using the MetCoOp Ensemble Prediction System (MEPS). MetCoOp stands for *Meterological Cooperation on Operational Numeric Weather Prediction (NWP)* and is a cooperation between the Nordic countries Norway, Sweden, and Finland. MET Norway produces weather forecasts from the MEPS system four times a day, respectively using data collected up to 00:00, 06:00, 12:00, and 18:00. The forecasts have a horizon of 67 hours and a horizontal resolution of 2.5 km and 65 vertical levels. The model runs on a Nordic domain consisting of 900 points in the zonal direction and 960 points in the meridional direction (Frogner, Singleton, Kø ltzow, & Andrae, 2019).

The MEPS data for this study are taken at 80 agl., i.e. hub height, and were collected and organized by Yngve Birkelund, professor at UiT The Arctic University of Tromsø. The weather variables of interest are zonal and meridional wind components (U, V), air temperature two meters above surface level T, and air pressure at surface level P. Two additional features are constructed from the wind components, namely wind speed ws and meteorological wind direction wd. For this analysis, meteorological direction is the most advantageous as it gives the direction the wind is blowing from which is essential when analyzing wind resources for wind power production. The reason for adding wind speed and direction is due to their easy interpretation during data analysis. Whether to use wind components or wind speed and direction during training and testing of the models will be clearer after data analysis. Wind speed and meteorological wind direction are calculated as follows

$$\begin{aligned} ws &= \sqrt{U^2 + V^2} \\ wd &= \frac{180}{\pi} \arctan 2(-U, -V). \end{aligned} \tag{21}$$

Forecasting wind power for the day-ahead market leaves the 06:00 weather forecast as the last available before the bidding deadline. Therefore, the day-ahead dataset is constructed using NWP data from the 06:00 forecast of the previous day. In contrast, for intraday bidding, the latest possible submission is one hour before the production hour. Although any of the four available forecasts can theoretically be used, the intraday dataset only incorporates NWP data from the 00:00 forecast of the current day.

9.3.3 On-site weather measurements

On-site weather measurements were considered to compare and analyze the accuracy of the MEPS weather forecasts. The data was collected from Met Norways Frost API where historical weather measurements are freely available. The measurement station lies on the shore at Vannøya, about 500 meters from Fakken wind park. Among the relevant features in this study, the station measures wind speed, meteorological wind direction, and air temperature.

In addition to being used for comparison reasons, the measured wind speed will also be tested as an exogenous variable in the form of a time-varying unknown real value. The reasoning behind this is to see if the models learn to map the difference in modeled and measured wind speed. During prediction, the model will only have access to the measured wind speed, and it is, therefore, relevant for the model to know this mapping.

10 Data Preprocessing

Raw data is data extracted from real-world processes without being changed in any way. Such data can contain inconsistencies from the real case due to faulty measuring equipment, wrong handling or human errors. Inconsistencies often come in the form of missing, duplicate, or simply just incorrect entries and are commonly termed dirty data. Using dirty data as input to a forecasting model can greatly diminish performance and it is therefore an important step to preprocess data before starting the data mining process. The following sections will go through data preprocessing steps to be performed on the day-ahead and intraday datasets. The steps are common to data preprocessing and thoroughly discussed in (García, Luengo, & Herrera, 2015), from which most of the information in this section is drawn.

10.1 Data exploration

Data exploration is the act of analyzing the raw data to gain insight into the data-generating processes. Such insight can make training the models and interpreting their results easier, reducing the training effort.

Sec. 12 presents the data exploration process done in this study and includes visual inspection, time series analysis, and exploration of the wind resources at Fakken wind park. The data exploration section will make the extent of preprocessing clearer.

10.2 Data cleaning

It is commonly said that the forecasting model is no better than the data it is given and the consensus is that the quality of the results is highly dependent on the quality of the input (Gudivada, Apon, & Ding, 2017). Data cleaning is the process of removing and treating dirty data for possible anomalies from the underlying distribution.

Finding such anomalies is therefore essential when cleaning data. Here it is beneficial to have a good understanding of the data that is being cleaned. A simple starting point is to detect outliers visually by looking at plots of the data, which in this study is a part of the exploratory analysis. Statistical tests are a more complicated option and can be used to detect outliers in the data that are unlikely to have been generated by the data-generating process. Whether to use statistical tests is generally clear after visual inspections of the data.

Scatter plots were used in this study to inspect possible anomalies. Two scatter plots were made and are illustrated in Fig. 16. The intentions with the plots are to both investigate outliers and to get insight into the wind resources at Fakken wind park. Outlier detection will be discussed here, while the wind resource analysis will be discussed in sec. 12.3.

Obvious anomalies in wind power production are when the park has downtime due to maintenance or other events. Downtime can be seen in scatter plots of power production

vs. wind speed as points with low to zero production while wind speeds lie between rated and cut-off wind speed. Even though such values could be removed, it is important to be aware of other reasons for low production, e.g. the wind direction. The plot (b) in Fig. 16 takes into account both wind speed and direction. When analyzing the two figures together, no clear anomalies are detected. For more accurate results, on-site measured weather parameters should be used in anomaly detection.

10.3 Missing values imputation

While some models for time-series data can naturally handle missing values (Mikalsen, Bianchi, Soguero-Ruiz, & Jenssen, 2018; Bianchi, Livi, Mikalsen, Kampffmeyer, & Jenssen, 2019), most forecasting models assume that the data is complete and it is, therefore, necessary to fill in missing values using imputation techniques. The question is what is the best method to avoid bias and to avoid missing important information? In some data-generating processes the missing values can have been introduced by specific sub-processes and wrong handling of missing values can therefore introduce strong biases.

Usually, there are two options: fill in missing values by a filling rule or discard them altogether. Filling in missing values can use other information or statistical tests to calculate the most likely value, or use padding or interpolation between the surrounding values. However, one must be careful when filling in values to not introduce information leakage from using future values in the imputation techniques. Removing missing values is another popular choice, but can be problematic if it means consistently removing data generated by a specific sub-process. Therefore, the data-generating process is an important factor to consider when dealing with missing values.

In this study, missing values can be found in the MEPS weather forecasts where a whole 'run', i.e. the 06:00 forecast on a specific day, is missing. This means that where there are missing values, a whole day of data is missing. Filling the values by padding or interpolating is then unfortunate since weather features are generally very volatile and filling a whole day with one value is not a good choice when taking the data-generating process into account. However, simply removing the values can result in breaking some temporal dependencies in the data. Deciding between these two rather undesirable options, padding the missing values with the last available value seems to be the most favorable option. This is due to removing the values that will introduce noise in the training process as wrong values will be used as a look-back window and forecast horizon.

10.4 Data splitting

The purpose of forecasting models is to predict well on unseen data, which was discussed in sec. 6.2. An important step to achieve this is to hold off a certain amount of data for validating and testing the model, while the largest part is used to fit the model to the data. For this study, the data is split into training, validation, and testing datasets. When determining the splitting ratio there is a trade-off taking place between the performance of training and evaluation. Deep learning models, compared to statistical models, usually require a lot of training data to perform well. Furthermore, it is important that the data in the training set is representative of the overall data for the model to be able to generalize well as discussed in sec. 6.2. This can be analyzed by inspecting the data distributions and variability of the proposed data splits. If there are large variations between the training, validation, and testing datasets, a different dataset split should be considered.

10.5 Feature scaling

Raw data usually contains large differences between the maximum and minimum values. In deep learning cases using stochastic gradient descent, this can result in unstable updating of the gradient and slow convergence. Additionally, it is also beneficial for the different features to be in the same range such that e.g. features with large values are not prioritized over features with small values.

This study uses Min-Max standardization to individually normalize each variable in the range $[0, 1]$ using the formula

$$x' = \frac{x - min(x)}{max(x) - min(x)}. \tag{22}$$

The scaler is fitted on the training data and used to normalize both the validation and testing data. This is done to avoid information leakage during training since it is important that the data used during testing remains unseen for the model.

11 Wind power forecasting models

For this thesis four models are tested, one main model and three baseline models. Out of the four models, three are deep learning models which come 'out of the box' from the forecasting library Pytorch Forecasting (Beitner, 2020) and one statistical model.

The main model called the *Temporal Fusion Transformer*, is based on the Transformer architecture and has achieved state-of-the-art results on the time series forecasting. This model was chosen for its reported ability to handle both short-term and long-term dependencies, as well as its ability to provide interpretable results through the use of an attention matrix. For baselines, previously popular models such as DeepAR, and ARIMA are chosen to investigate the results of different architectures for time series forecasting.

The next subsections will present each of these models, with a special focus on the Temporal Fusion Transformer.

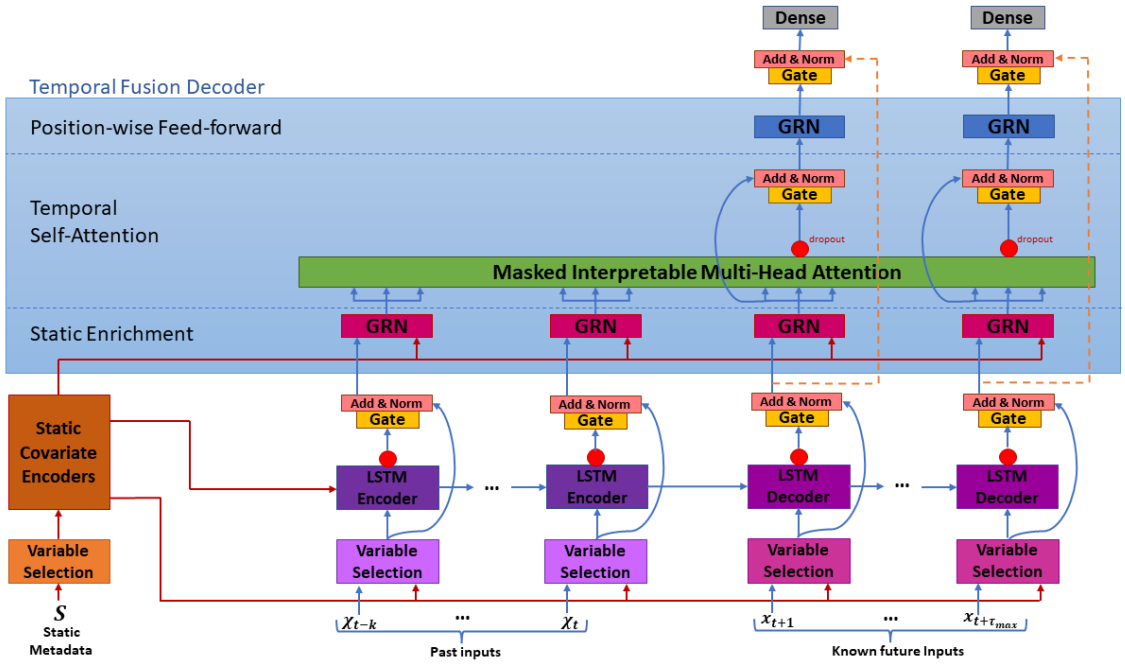


Figure 9: Architectural overview of the Temporal Fusion Transformer

11.1 Temporal Fusion Transformers for Interpretable Multi-Horizon Time Series Forecasting

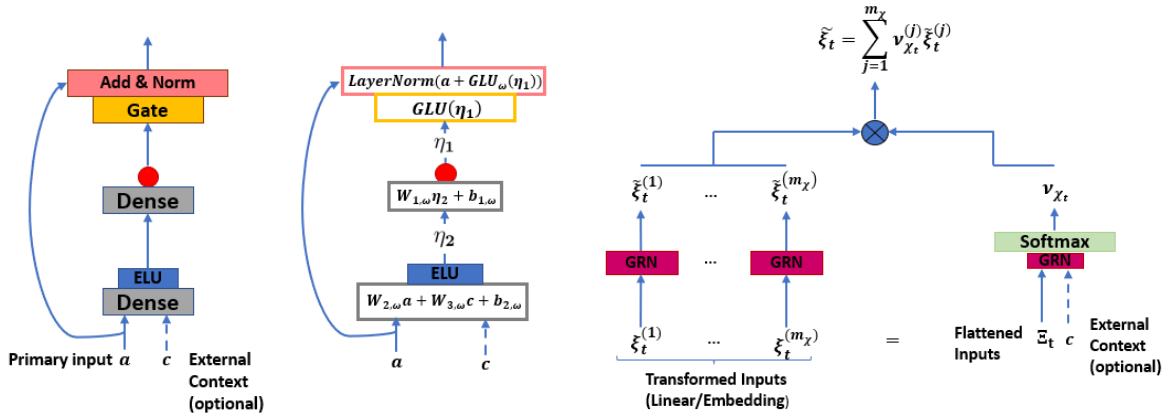
The Temporal Fusion Transformer (Lim, Arik, Loeff, & Pfister, 2020), TFT, is an adaptation of the original Transformer model to accommodate multi-horizon time series forecasting. TFT combines both sequence-to-sequence modeling and the Transformer into one, hence the name Temporal Fusion Transformer.

The TFT has the option to include known future inputs, exogenous time series and static metadata as covariates. The inclusion of static data means you could use the same model on different time series with similar characteristics. A chain of stores could collect data from all its stores and apply the same model to all.

The TFT creates multi-horizon forecasts using the MIMO method, meaning that it takes the whole lookback window as input and outputs predictions for the whole forecasting horizon T.

11.1.1 Model design

The model is an encoder-decoder structure as is typical for sequence models. This is where the TFT mixes Transformers and seq2seq models with an LSTM-based encoder and Attention-based decoder. The reasoning is to leverage the LSTM's ability to map local dependencies with the Attention mechanism's ability to map long-distance dependencies. In addition, the seq2seq layer will substitute the positional encoding in the original Transformer. Some building blocks are used throughout the whole model and it is beneficial to introduce these before going into the encoder-decoder structure.



(a) Gated Residual Network (GRN) (b) Variable Selection Network

Figure 10: Building blocks in the Temporal Fusion Transformer network

Gated Residual Network - GRN

The Gated Residual Network, GRN, gives the model the opportunity to apply non-linear processing. At its core, it is a two-layer fully connected (FC) network, however, the GRN contains an additional layer which gives the possibility to skip the non-linear processing altogether. This is done by a Gated Linear Unit, GLU, which employs the sigmoid activation function to suppress unnecessary parts of the network.
The GRN takes in two inputs, one primary input $\mathbf{a}$ and one optional external context $\mathbf{c}$, and is summarized by the following equations:

$$\begin{aligned} GRN_\omega(\mathbf{a},\mathbf{c}) &= LayerNorm(\mathbf{a} + GLU_\omega(\eta_1)) \\ \eta_1 &= \mathbf{W_{1,\omega}}\eta_2 + \mathbf{b_{1,\omega}} \\ \eta_2 &= ELU(\mathbf{W_{2,\omega}a} + \mathbf{W_{3,\omega}c} + \mathbf{b_{d,\omega}}) \\ GLU_\omega(\gamma) &= \sigma(\mathbf{W_{4,\omega}}\gamma + \mathbf{b_{4,\omega}}) \odot (\mathbf{W_{5,\omega}}\gamma + \mathbf{b_{5,\omega}}) \end{aligned} \tag{23}$$

Where η_1, η_2 represents the FC network and ELU is the Exponential Linear Unit. The learnable parameters have the subscript $(\cdot)_\omega$ which denotes weight sharing. The gated residual connection is represented by eq. 3.12 and 3.15. In 3.12, the inclusion of GLU gives the possibility to control how much the non-linear processing should affect the output. Potentially, the outputs of the GLU can all be close to 0 which will skip the layer altogether.

Variable Selection Network

The prediction problem contains multiple inputs, but not all are relevant at every point. The TFT, therefore, has taken use of Variable Selection Networks to provide instance-wise variable selection on both the static and time-dependent covariates. This part of the architecture has a dual purpose, it gives insight into significant variables for the prediction, while also removing noisy inputs that have poor impact on prediction performance. It is common for time-series datasets to contain both noise and varying degrees of salient features. By letting the model learn which variables are most salient, the model both increases accuracy by pruning unnecessary input and learns valuable insight into trends and other significant events.

At a basic level, the Variable Selection Networks assign Softmax-weights, termed variable

selection weights, to the input sequences to suppress unimportant input. Each input type is assigned its own Variable Selection Network, with its own learnable parameters.

The input is first embedded. Categorical variables use entity embeddings, while continuous variables are linearly transformed. Similarly as in 'Attention is All you need' (Vaswani et al., 2017) it is important that all variables have the same dimension due to the residual connections. All variables are therefore transformed to a d_{model} dimensional vector denoted $\xi_{\mathbf{t}}^{(\mathbf{j})} \in \mathbb{R}^{d_{model}}$, where j represents the jth variable at time t. These vectors are processed in two separate parts of the Variable Selection Network. One part calculates feature vectors, while the other part calculates the variable selection weights. To calculate feature vectors, the transformed input is further non-linear and processed by their own GRN, meaning each variable has a GRN with weights shared across all time steps t.

$$\tilde{\xi}_{\mathbf{t}}^{(\mathbf{j})} = GRN_{\tilde{\xi}(\mathbf{j})}(\xi_{\mathbf{t}}^{(\mathbf{j})}) \tag{24}$$

Where $\tilde{\xi}_t^{(j)}$ is the processed feature vector for variable j. To calculate variable selection weights, each of the transformed inputs for time step t is flattened and collected.

$$\mathbf{\Xi_t} = \left[\xi_{\mathbf{t}}^{(\mathbf{1})^{\mathbf{T}}}, \ldots, \xi_{\mathbf{t}}^{(\mathbf{m}_\chi)^{\mathbf{T}}}\right]^T \tag{25}$$

This flattened vector is also non-linearly processed by a GRN, together with the optional context vector, and the output is brought through a Softmax layer to produce the variable selection weights.

$$\nu_{\chi_{\mathbf{t}}} = Softmax(GRN_{\nu_\chi}(\mathbf{\Xi_t}, \mathbf{c_s})) \tag{26}$$

The feature vectors and the weights are combined:

$$\tilde{\xi}_{\mathbf{t}} = \sum_{j=1}^{m_\chi} \nu_{\chi_t}^{(j)} \tilde{\xi}_{\mathbf{t}}^{(\mathbf{j})} \tag{27}$$

Interpretable Multi-Head Attention
The underlying Attention mechanism is the same, a scaling of Values given the relationship between Queries and Keys. The authors have, however, modified the original Multi-head Attention in two ways.

1. Instead of one Value matrix for each Attention head, they use one Value matrix for all Attention heads

2. Instead of concatenating the output matrices, they sum them.

They call the modified Multi-head Attention for Interpretable Multi-head Attention, and the changes are expressed as follows:

$$\begin{aligned} \text{InterpretableMultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) &= \mathbf{\tilde{H}W_h} \\ \text{where } \mathbf{\tilde{H}} = 1/H \sum_{h=1}^{m_H} \text{Attention}(\mathbf{QW_Q^{(h)}}, \mathbf{KW_K^{(h)}}, \mathbf{VW_V}) \end{aligned} \tag{28}$$

Where:

$$\mathbf{Q} \in \mathbb{R}^{N \times d_{attn}}, \quad \mathbf{K} \in \mathbb{R}^{N \times d_{attn}}, \quad \mathbf{V} \in \mathbb{R}^{N \times d_V},$$

and

$$\mathbf{W}_Q^{(h)} \in \mathbb{R}^{d_{model} \times d_{attn}}, \quad \mathbf{W}_K^{(h)} \in \mathbb{R}^{d_{model} \times d_{attn}}, \quad \mathbf{W}_V \in \mathbb{R}^{d_{model} \times d_V},$$
$$\mathbf{W}_H \in \mathbb{R}^{d_{attn} \times d_{model}}$$

The dimensions of the matrices, $d_V = d_{attn} = d_{model}/m_H$, are chosen by the number of Attention heads, m_H.
The final weight matrix, $\mathbf{W}_h$, is used as a linear mapping.

11.1.2 Encoder

The encoder consists of two parts, a Static Covariate Encoder to process the static variables and a seq2seq layer to process the continuous variables.

11.1.2.1 Static Covariate Encoder

First, static metadata is sent through its own variable selection network to produce static features which are then passed on. The static covariate encoder contains four different GRNs which produce four different context vectors, $\mathbf{c_s}, \mathbf{c_e}, \mathbf{c_c}, \mathbf{c_h}$. The context vectors are sent to different parts of the model and give information about the static metadata at multiple points in the network. As to where the context vectors are sent:

$\mathbf{c_s} \longrightarrow$ Past and future input Variable Selection Network
$\mathbf{c_e} \longrightarrow$ Static Enrichment layer (TFT Decoder)
$\mathbf{c_c} \longrightarrow$ Initiate cell state in seq2seq layer
$\mathbf{c_h} \longrightarrow$ Initiate hidden state in seq2seq layer

11.1.2.2 Sequence-to-Sequence Layer

The authors propose to leverage both seq2seq layers and an Attention layer to improve the performance of the Transformer, while simultaneously replacing the positional encoding used in the original Transformer. When arriving at the Attention layer, the sequence is already encoded with pattern information beyond that of the positional encoding alone.

Before being sent to the seq2seq layer, the past and future input are passed through variable selection networks yielding $\tilde{\xi}_{\mathbf{t-\tau:t}}$ for past values and $\tilde{\xi}_{\mathbf{t+1:t+T}}$ for future values. The seq2seq layer consists of LSTM encoder and decoder blocks, where the observed (past) values is input to the encoder blocks and the known future values are input to the decoder block. This means that there are τ (length of look-back window) LSTM encoder blocks, and T (length of forecast window) decoder blocks. In addition, context vectors are passed to initiate both the cell state and the hidden state in the first LSTM encoder block. The seq2seq layer generates a set of uniform temporal features $\phi(t, n) = \{\phi(t, -\tau), \ldots, \phi(t, T)\}$ that is sent as input to the Temporal fusion decoder after a gated residual connection yielding $\tilde{\phi}(t, n) = LayerNorm(\tilde{\xi}_{\mathbf{t+n}} + GLU_\phi(\phi(t, n)))$. Where $n \in [-\tau, T]$ is a position index.

11.1.3 Decoder

The Temporal Fusion Decoder consists of three layers. A static Enrichment Layer encodes extra static information, and a self-Attention layer is succeeded by an FC layer. In addition, the whole decoder is succeeded by a gated residual connection meaning the decoder can be skipped entirely if the additional complexity is not necessary.

Static Enrichment
Before arriving at the Attention layer, the uniform sequence $\tilde{\phi}(t,n)$ from the seq2seq layer is encoded with additional static information. This is done by sending the sequence and context vector $\mathbf{c_e}$ to GRN blocks which share weights across the whole layer:

$$\theta(t,n) = GRN_\theta(\tilde{\phi}(t,n), \mathbf{c_e}) \tag{29}$$

Temporal Self-Attention
The static enriched features are stacked into one self-Attention input matrix, $\Theta(t) = \left[\theta(t,-\tau), \dots \theta(t,T)\right]^T$ before entering the Temporal Self-Attention layer. Interpretable Multi-Head Attention is applied as follows:

$$\mathbf{B}(t) = \text{InterpretableMultiHead}(\mathbf{\Theta}(t), \mathbf{\Theta}(t), \mathbf{\Theta}(t)) \tag{30}$$

Which results in a new vector:

$$\mathbf{B}(t) = [\beta(t,-\tau), \dots, \beta(t,T)] \tag{31}$$

Succeeding points are masked, such that the prediction only build on preceding features. The Self-Attention layer is followed by a gated residual connection which yields:

$$\delta(t,n) = \text{LayerNorm}(\theta(t,n) + GLU_\delta(\beta(t,n))) \tag{32}$$

Position-wise Feed-forward
The final layer of the decoder is a Feed-forward network executed by GRNs and gives:

$$\psi(t,n) = GRN_\psi(\delta(t,n)) \tag{33}$$

The whole decoder is succeeded by a gated residual connection to the seq2seq layer:

$$\tilde{\psi}(t,n) = \text{LayerNorm}(\tilde{\phi}(t,n) + GLU_{\tilde{\psi}}(\psi(t,n))) \tag{34}$$

Quantile outputs
On top of point forecasts, the authors also generate quantile forecasts to give additional information of the uncertainty of the forecast. This is done by a final linear transformation of the decoder output in a FC layer.

$$\hat{y}(q,t,T) = \mathbf{W_q}\tilde{\psi}(t,T) + \mathbf{b_q} \tag{35}$$

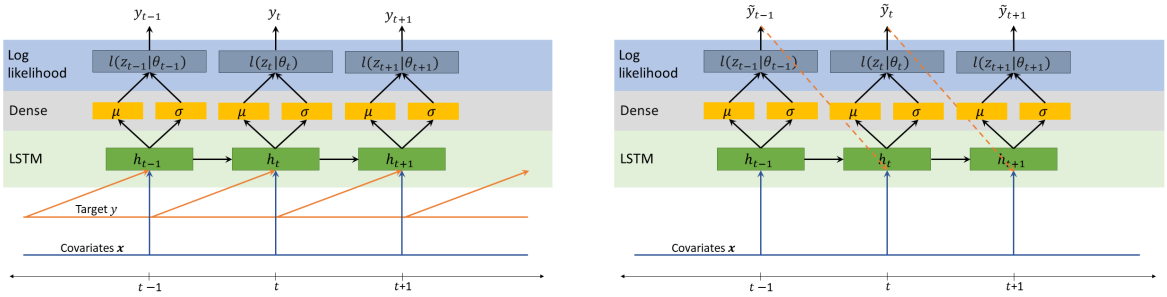
Where $\mathbf{W_q} \in \mathbb{R}^{1\times d_{model}}$ and $\mathbf{b_q} \in \mathbb{R}$ are weight and bias connected to the quantile q.

Training
TFT produces probabilistic output by QR as described above. The model is therefore trained using quantile loss summed across all quantile outputs and the whole forecast horizon as described in (Lim et al., 2020) with the following function.

$$\mathcal{L}(\Omega, \mathbf{W}) = \sum_{y_t \in \Omega} \sum_{q \in \mathcal{Q}} \sum_{T=1}^{T_{max}} \frac{QL(y_t, \hat{y}(q, t-T, T), q)}{NT_{max}} \tag{36}$$

Where QL stands for the quantile loss function and is defined as in sec. 4.4 in the theory. The loss for each training sample $y_t \in \Omega$ is summed and averaged to get the loss for the whole epoch.

Regularization
Several regularization techniques are used, therein early stopping, dropout, gradient clipping, and weight decay. Early stopping is implemented similarly for all models and stops the training if the validation loss has not improved within a small threshold in the last 50 epochs. Dropout in TFT is implemented in various parts of its architecture, therein at each fully connected layer present in the GRN's, GLU's and the Add&Norm gates. Furthermore, if the model uses multiple LSTM layers there is also a dropout between these.



(a) DeepAR training procedure

(b) DeepAR testing procedure

Figure 11: DeepAR model illustration

11.2 Baselines

11.2.1 DeepAR

DeepAR by (Salinas et al., 2020) is an LSTM-based neural network producing probabilistic forecasts using Bayesian inference. Amazon developed the model to forecast retail demand and their intention was to build one model that can generalize to different product demands since most demand time series have common characteristics. The network then originally have a learned scaling layer which will not be used in this study. In theory, one could use this network to forecast production for different wind parks or electrical demand for different households or cities.

Model design
DeepAR aims to model the conditional distribution of the future of time series y given the past values and covariates $\mathbf{x}$

$$P(y_{t_0:T}|y_{1:t_0-1}, \mathbf{x_{1:T}}) \tag{37}$$

Here it is important to note that the covariates are of a known nature, e.g. DateTime covariates or in the case of this study: NWP data. As described in sec. 4.4.2, it is assumed that the model distribution is a product of, in this case, Gaussian likelihood factors.

$$P(y_{t_0:T}|y_{1:t_0-1}, \mathbf{x_{1:T}}) = \prod_{t=t_0}^{T} l_G(y_t|\mu, \sigma) \tag{38}$$

The likelihood function is parameterized by the output of the network at each time point:

$$\begin{aligned} \mu(\mathbf{h_t}) &= \mathbf{w}_\mu^T \mathbf{h_t} + b_\mu \\ \sigma(\mathbf{h_t}) &= \log(1 + exp(\mathbf{w}_\sigma^T \mathbf{h_t} + b_\sigma)) \\ \mathbf{h_t} &= h(\mathbf{h_{t-1}}, y_{t-1}, \mathbf{x}_t, \mathbf{\Theta}) \end{aligned} \tag{39}$$

Where $\mathbf{h_t}$ is the output of the LSTM network with network parameters $\mathbf{\Theta}$ and μ, σ is the output of two parallel fully connected networks. The transformation $log(1 + exp(\cdot))$ corresponds to a Softplus activation and is done to avoid a negative standard deviation. During training, the previous target time step is fed as input to the model, while during

testing a sample from the predicted likelihood function of the previous time step is used as input. Both processes are depicted in Fig. 11.

Information is passed from the look-back window to the prediction horizon using a seq2seq setup as described in sec. 7.1.3. The difference from an ordinary seq2seq network is that the encoder and decoder in this case use the same architecture and share weight. The authors report that this resulted in the best results and justifies it by encoding and decoding on the same type of data.

Training
DeepAR uses Bayesian inference, and the loss function is, therefore, the negative log-likelihood of the distribution it is modeling. For DeepAR that distribution is the Gaussian distribution parameterized by $\theta(\mathbf{h_t})$:

$$\begin{aligned} \mathcal{L} &= \sum_{t=t_0}^{T} \log l_G(y_t|\theta(\mathbf{h_t})) \\ l_G(y_t|\theta(\mathbf{h_t})) &= (2\pi\sigma(\mathbf{h_t})^2)^{-\frac{1}{2}} exp(-(z-\mu(\mathbf{h_t}))^2/(2\sigma(\mathbf{h_t})^2)) \end{aligned} \tag{40}$$

Where $\mathbf{h_t}$ is the output of the LSTM network and the parameters σ and μ are the linear transformations computed in eqs. 39.

Regularization
Similarly, as with TFT, DeepAR uses early stopping, dropout, gradient clipping, and weight decay as regularization techniques. Dropout is only implemented in the LSTM layer(s), and is only used if there is more than one LSTM layer. Early stopping, gradient clipping and weight decay is implemented in the same manner as TFT.

11.2.2 ARIMA

Autoregressive (AR) Model
An autoregressive model, $AR(p)$, is based on the assumption that current and future values of a time series are dependent on past values up to a specified number, p. This number can be determined by inspecting the partial autocorrelation function (PACF) which will give information on a time series memory as introduced in sec. 4.3.

The AR model exists both in univariate form and can be extended to multivariate time series with the vector autoregressive model, $VAR(p)$ as follows (Tsay, 2014):

$$\mathbf{y_t} = \phi_\mathbf{0} + \mathbf{\Phi} \mathbf{y_{t-p:t-1}} + \mathbf{a_t} \tag{41}$$

Where $\mathbf{\Phi}$ is a vector of the model parameters, $\phi_\mathbf{0}$ is a constant vector and $\mathbf{a_t}$ is an error vector of stochastic variables with zero mean vector and positive -definite covariance matrix.

Moving Average (MA) Model
The moving average model, $MA(q)$, is based on the assumption that current and future values of time series are dependent on the model's past forecasting errors, $\mathbf{a_t}$, up to a user-specified number, q. The number can be determined by inspecting the autocorrelation function (ACF). Similarly, as with the autoregressive model, the moving average model also has a multivariate extension, $VMA(q)$ (Tsay, 2014):

$$\mathbf{y_t} = \theta_\mathbf{0} + \mathbf{\Theta}(\mathbf{B})\mathbf{a_{t-p:t-1}} \tag{42}$$

where θ_0 is a constant vector equal to the mean of $\mathbf{y_t}$ and $\mathbf{\Theta}(\mathbf{B})$ is the MA matrix polynomial in the back-shift operator B.

Autoregressive Integrated Moving Average (ARIMA) Model
A common choice for time series forecasting has historically been the autoregressive moving average model, ARMA(p,q), which is a combination of the AR(p) and MA(q) models. ARMA(p,q) then models the linear combination of both the past values and the past errors. AR(p), MA(q), and therefore also ARMA(p,q) requires the time series to be stationary because it is easier to model due to stable statistical properties. Therefore alternate models have been developed to adapt to time series showing trend and seasonality, respectively autoregressive integrated moving average, ARIMA(p,d,q), and seasonal ARIMA, SARIMA(p,d,q)×(P,D,Q)m.

The ARIMA model deals with the trend and non-seasonality by applying differencing to the time series before fitting the ARMA model on it. The parameter, d, denotes the number of times the time series has been differenced and a low number is usually preferred as important information can be lost when differencing a time series too many times. The SARIMA integrates seasonal parameters (P,D,Q)m, where P,D, and Q refer to the seasonal alternatives for p,d,q, and m refers to the seasonality of the series, which for hourly data usually is 24.

Similarly, as with the MA and AR models, the ARIMA model has a multivariate extension, which in this case is called ARIMAX (Mehandzhiyski, 2023):

$$\Delta\mathbf{y_t} = \mathbf{c} + \beta\mathbf{X} + \mathbf{\Phi}\mathbf{y_{t-p:t-1}} + \mathbf{\Theta}(\mathbf{B})\mathbf{a_{t-p:t-1}} + \mathbf{a_t} \tag{43}$$

Where the constant c is the additive combination of the constants ϕ_0, θ_0 from the AR and MA expressions and the term $\beta\mathbf{X}$ denotes the exogenous variables along with its coefficients.

Probabilistic Forecasting

Probabilistic forecasts from ARIMA models are obtained by using the model residuals to compute prediction intervals (PIs) which were introduced in sec. 4.4.1. The PIs are given by

$$\hat{y_{t+T}} \pm c_\alpha \hat{\sigma}_h \tag{44}$$

where $\hat{y}_{t+T}$ is the point prediction at time T, c_α is a confidence parameter indicating the significance level of the PI, and $\hat{\sigma_h}$ is an estimate of the standard deviation of the forecasting distribution derived from the model residuals (R. J. Hyndman & Athanasopoulos, 2018). Multi-step forecasting tends to give wide prediction intervals since the standard deviation of the residuals and the true distribution tend to inflate as the forecasting horizon increases (Brockwell, Brockwell, Davis, & Davis, 2016). This is natural, as the prediction errors accumulate at every time step. Autoregressive models, such as ARIMA and DeepAR, make multi-step predictions in a recursive manner as described in sec. 4.1. Meaning that the previous prediction is fed as input when predicting the current time step. This kind of multi-step forecasting is prone to error accumulation which for ARIMA models manifests itself not only in the point prediction but also in the width of the PI as the forecasting horizon increases.

Part IV / Experiments

12 Raw data analysis

It is important to have some knowledge of the data before processing them with the ML models because it makes tuning the model and interpreting the results more intuitive. In this section, the two different energy datasets are presented and the raw data is analyzed. The analysis should give an understanding of the properties of the time series and its relation to the exogenous features.

During the analysis, the year 2017 will be used to explore and analyze the properties seen in the data, and will be called the exploration dataset from here on. For the NWP data, the 00:00 forecast for the current day will be used when only testing one set of NWP data. This is because the 00:00 forecast is the more reliable one when choosing between the two NWP datasets which in this analysis is preferable.

12.1 Descriptive statistics

Tab. 1 presents descriptive statistics of the exploration datasets. The top panel includes wind power output and measured wind speed, which are common in both datasets and modeled weather features from the day-ahead 06:00 run and the intraday 00:00 run respectively to the left and the right. There appear to be small deviations between the two sets of modeled weather features. Both modeled wind speeds report lower wind speeds than the true measured value. The measured wind speed exhibit both higher mean, median and standard deviation implying that the measured wind is more volatile than the MEPS models are able to model. The features, $\hat{ws}, \hat{U}, \hat{P}, \hat{T}$ along with the endogenous variable, have lower median than mean values, implying a right-skewed distribution. Vice versa, the features, $\hat{wd}, \hat{V}$, have higher median than mean values, implying a left-skewed distribution.

Tab. 2 shows the linear correlations between park power output and the exogenous variables for both the day ahead and intraday datasets. The most significant value of the same variable between the two datasets is printed in bold. The intraday values are generally more correlated to the park power output than the day ahead values, with the exception of westward wind, U. An important takeaway from the table is the significantly higher correlation of wind speed, ws, compared to the decomposed wind, U and V. Due to this, it is preferred to use wind speed and direction rather than decomposed wind in the models. Additionally, it is interesting to see that temperature and pressure have such a low correlation implying that low pressure and low temperatures are correlated with high park power output and vice versa. It is therefore good reason to also keep these parameters as input to the WPF models.

dayahead/intraday	mean	std	min	median	max
Power output [MWh]	15.24	15.86	0	9.34	53.5
ws [m/s]	8.0	4.51	0	7.2	30.1
$\hat{ws}$ [m/s]	7.05/6.96	3.86/3.88	0.14/0.08	6.48/6.4	28.75/30.77
$\hat{wd}$ [°]	180/176	97.63/96.72	0.01/0.1	188/186.7	359.9/359.9
$\hat{U}$ [m/s]	0.39/0.35	5.28/5.27	-23.4/-25.3	0.3/0.24	20.92/21.77
$\hat{V}$ [m/s]	1.33/1.4	5.9/5.8	-23.4/-20.2	2.5/2.5	17.86/18.46
$\hat{P}$ [hPa]	1005/1002	12.8/12.86	967/966	1002/1002	1042/1042
$\hat{T}$ [K]	277/277	5.11/5.12	263/264	276.4/276.5	292/291

Table 1: Descriptive statistics of the exploration data.

data	$\hat{ws}$ [m/s]	$\hat{wd}$ [°]	$\hat{U}$ [m/s]	$\hat{V}$ [m/s]	$\hat{P}$ [Pa]	$\hat{T}$ [K]
Day ahead	0.63	0.045	**-0.018**	0.19	-0.16	-0.16
Intraday	**0.66**	0.045	-0.013	**0.21**	**-0.17**	**-0.17**

Table 2: Linear correlation between modeled weather features and wind power output

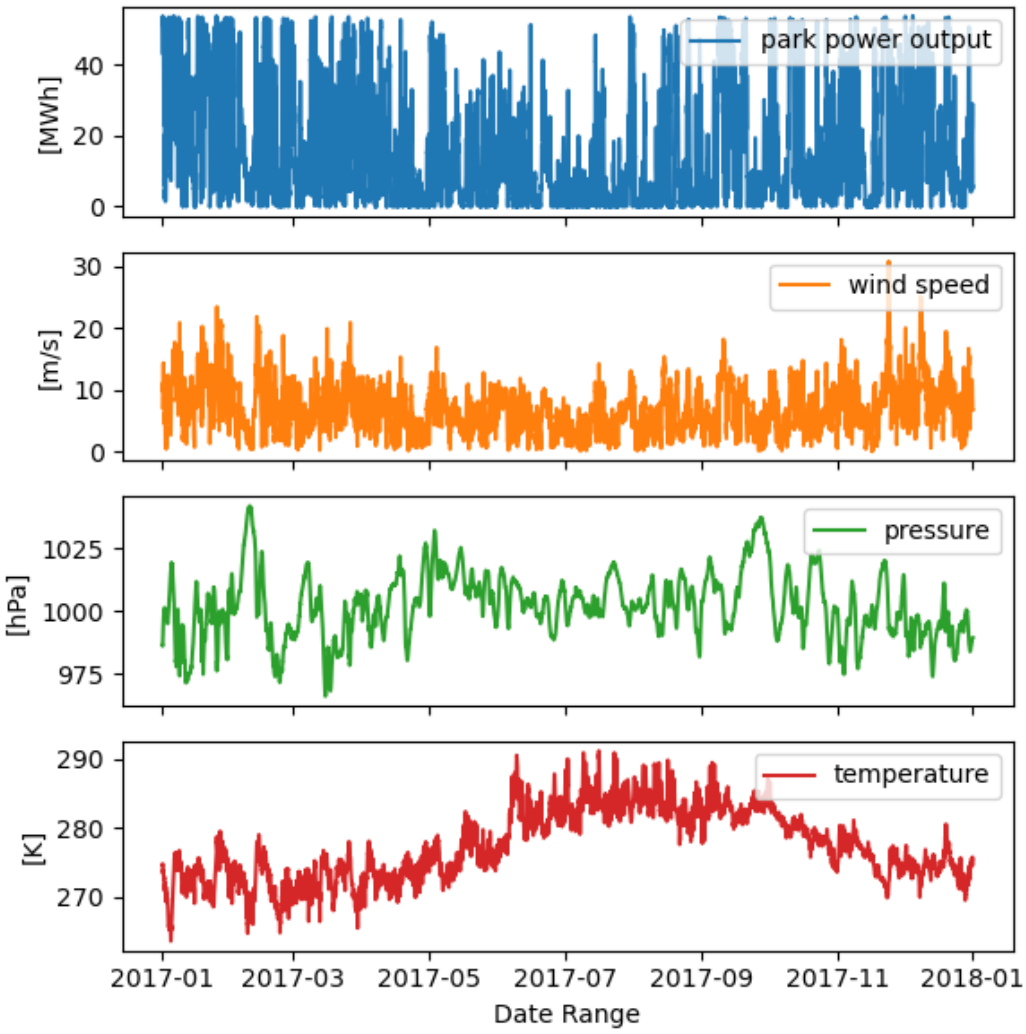


Figure 12: Time series plot over the year 2017 for park power output and modeled wind speed, pressure and temperature

12.2 Time series analysis

The next subsection will go through some basic time series analysis steps to get an understanding of the properties exhibited in the time series.

12.2.1 Visual inspection

Fig. 12 shows the time series of park power output, wind speed, pressure, and temperature for the year 2017. The time series exhibits some seasonal variations with lower variances in park power output and wind speed during summer months compared to winter months. Obviously, this seasonal variation can also be seen in the temperature time series with higher temperatures during summer. The pressure is also more stable during the summer months. Wind direction is not included in this figure which is due to its cyclic nature making it visually uninformative in this kind of time series plot.

12.2.2 Autocorellation and Partial Autocorrelation

The autocorrelation function (ACF) and the partial autocorrelation function (PACF) are commonly used plots in time series analysis to determine the order of AR and MA lags in the ARIMA model. Additionally, they also give insight to seasonality, memory, and if the time series is stationary or not. Specifically, the ACF of a time series with seasonal information will have repeating peaks at regular intervals. While a stationary time series will drop off to zero shortly after lag 0 (Shumway & Stoffer, 2017). The memory of a time series can be read from the PACF plot, which explains the linear dependencies that the ACF function can not explain by removing the intermediate linear dependencies. In other words, at e.g. lag 2 only the dependency between x_t and x_{t-2} is investigated, removing the forward and backward dependencies from x_{t-1} (Shumway & Stoffer, 2017). Significant values in the PACF show to which extent the time series has memory. It is therefore a valuable plot when determining the look-back window discussed in Sec. 4.3.

Fig. 13 shows the autocorrelation and the partial autocorrelation of the Fakken wind power output time series with 100 lags. It is clear from the PACF plot that there are little to no linear dependencies between the lags. It shows that the PACF cuts off at lag 2, meaning that it is unlikely to find a dependency that goes beyond two time steps. If there exist non-linear dependencies those are not captured by this method, which is linear. The ACF tails off to zero and is not showing any signs of short-term seasonality. A tailing off ACF and PACF cutting off at lag 2 indicates that this model is an AR(2).

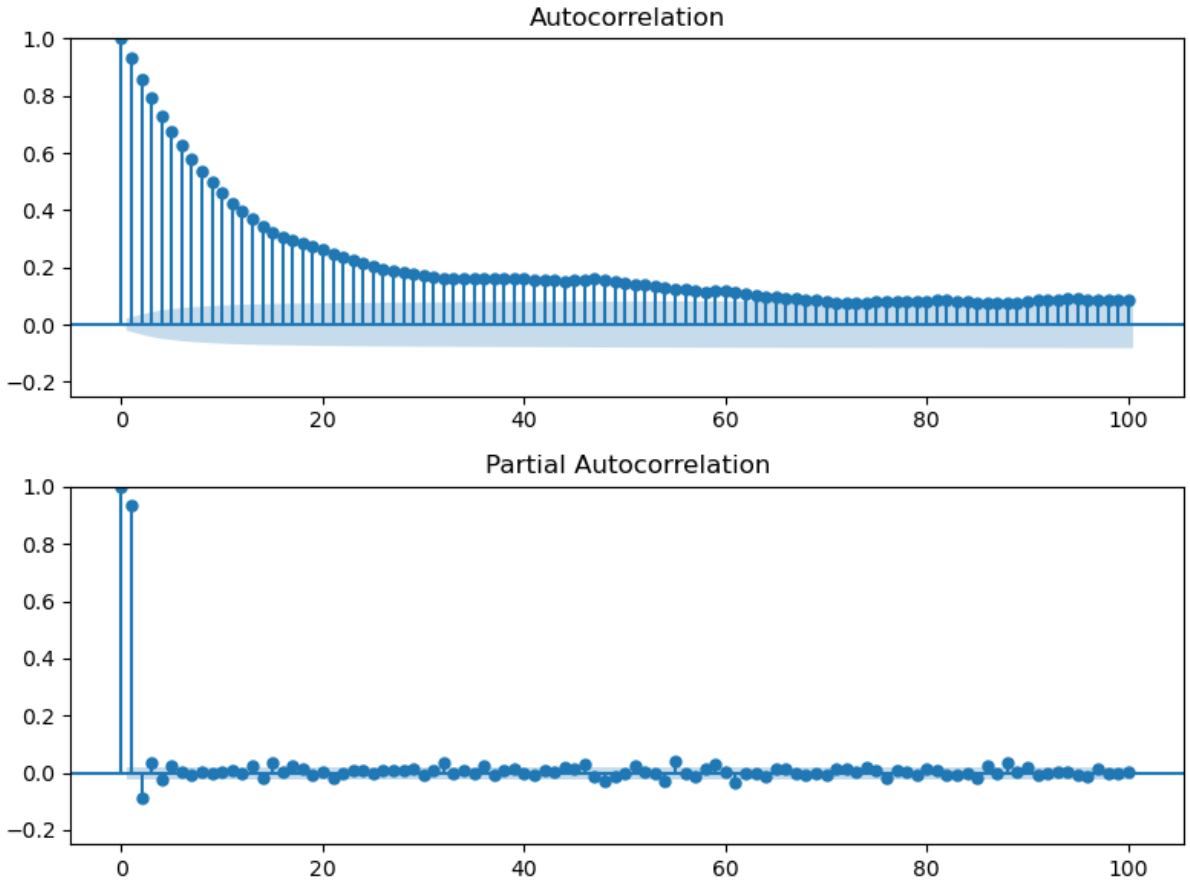
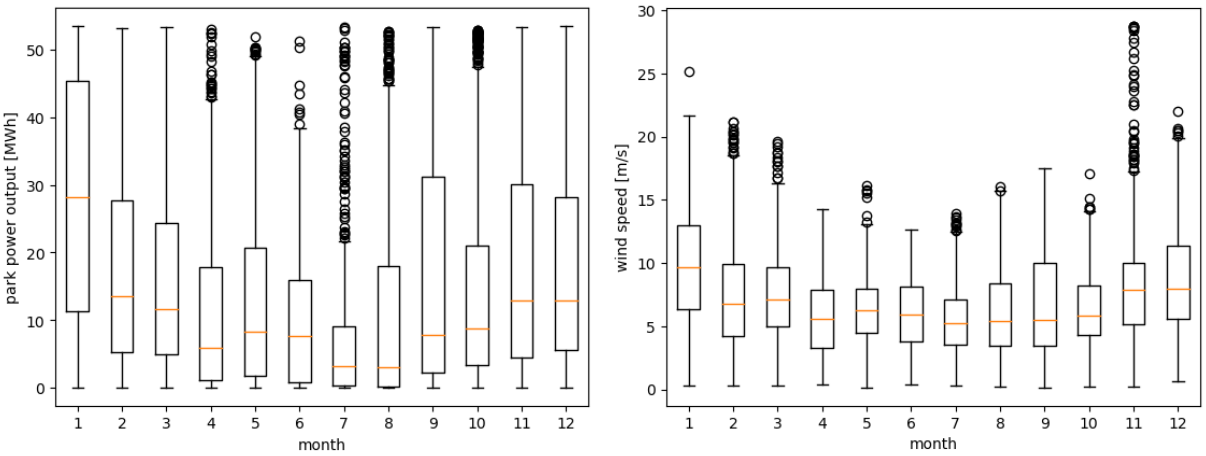


Figure 13: Autocorrelation and partial autocorrelation of park power output

12.2.3 Seasonal analysis

In fig. 12 there were signs of seasonal tendencies, giving reason to investigate if there are clear seasonal variations in the park power output. Seasonal variations are here presented using boxplots in Fig. 14, where monthly variations are plotted for both park power output and MEPS wind speed. Boxplots are a good visualization of the distribution of a dataset and percentiles are presented using boxes and whiskers. The box extends from the 25th percentile to the 75th percentile, while the whiskers present the 0th and 100th percentile. The box is divided into two parts where the line separating them represents the median of data. Points plotted outside of the whiskers are called outliers. Fig. 14a shows that Fakken wind park produces in the whole range, from 0 to 54 MWh for all months. However, there is a clear 'drop' during the summer months, both with lower variability and a lower median than for the winter months. Comparing fig. 14a and fig. 14b makes it clear that wind speed is the leading factor to wind power production. Increased variance in the wind data leads to increased variance in power park output. Additionally, the cubic relationship can be seen in the figures as the variance in power park output is larger than that in wind speed.



(a) Monthly variability of park power output (b) Monthly variability of wind speed

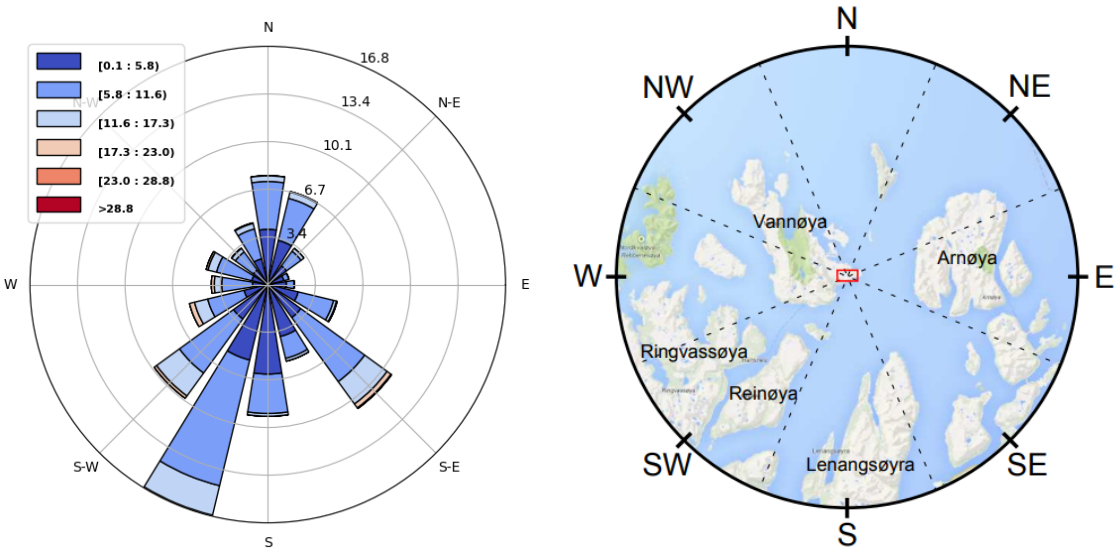
Figure 14: Monthly variability of park power output and wind speed in exploration dataset. Both figures (a) and (b) show a clear seasonal variability.

12.3 Wind and terrain

Fig. 15 shows two figures that give insight into the wind resources at Fakken wind park. Figure (a) shows a wind rose made from wind speed and wind direction of the MEPS NWP data for the years 2017-2019. A wind rose is a circular histogram indicating at what frequency the wind blows from a certain direction. Each bin is sectioned by wind speed such that the histograms give information on both frequency of direction and wind speed at a given direction. Fig. 15a suggests that the most prominent wind directions are from the South South-West (S-SW) and South East (SE) corresponding to degrees 200 and 135. Another less prominent, direction is from the North (N) corresponding to degree 0. Most interesting is the wind directions giving wind speeds in section $[11.6-17.3]$ corresponding to the wind turbines' rated wind speed. These sections are colored as the lightest blue in the wind rose. It is therefore expected that the directions SW, S-SW, and SE will result in the highest wind power production.

Fig. 15b by (Jacobsen, 2014) illustrates the location of Fakken (red box) with surrounding terrain and directions. The islands in the area are characterized by mountains and complex terrain creating natural highways for the wind to travel. The figure provides insight into the most prominent wind directions seen in Fig. 15a as the wind directions S-SW and SE correspond to the fjords between Reinøya, Lenangsøya, and Arnøya, while the wind direction N corresponds to the open North Sea.

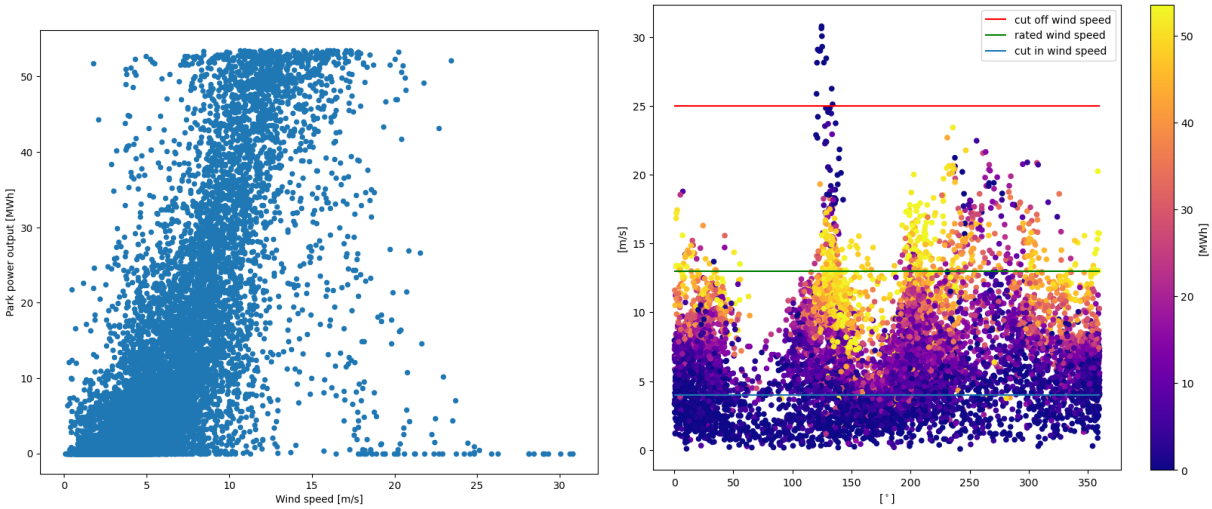
Fig. 16 shows scatter plots illustrating the relationship between wind power production, wind speed, and wind direction. Figure (a) shows the scatter plot between wind speed and power production resulting in a relationship similar to the power curve seen in Fig. 7. However, there are some inconsistencies with an ordinary power curve as delivered by the wind turbine producers resulting from the use of modeled wind data instead of on-sight weather measurements. Examples of this inconsistency are when there is high power production with low wind speeds and low power production with wind speeds in between rated and cut-off wind.



(a) Wind rose based on MEPS wind speed and direction during 2017-2019

(b) Location of Fakken wind park (Jacobsen, 2014).

Figure 15: Wind rose and location of Fakken Wind park.



(a) Park power output and MEPS wind speed scatter plot.

(b) wind speed and wind direction scatter plot colored by park power output

Figure 16: Scatter plots of power production, wind speed and wind direction

Fig. 16b shows the scatter plot of wind speed given wind direction colored by power production. This was done to illustrate at which wind speeds and wind directions result in the highest and lowest power production. Additionally, *cut in*, *rated*, and *cut off* wind speeds are plotted to give further explainability to the plot. The plot confirms the assumptions made from the wind rose as the degrees around 0, 135, and 220 are the ones resulting in the highest power production. Another key takeaway from the plot is that there are consistently higher winds than the MEPS model forecasts since both the rated and cut-off wind speeds are higher than the corresponding peak production and cut-off production.

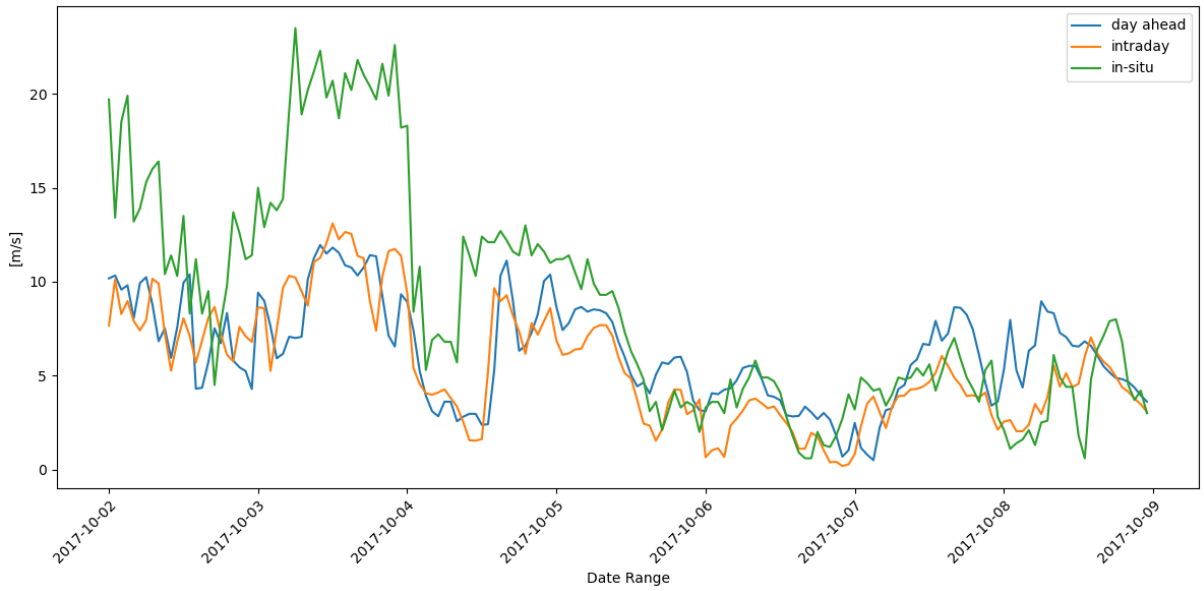


Figure 17: Comparison between the two wind speed forecasts and the measured wind speed at Fakken for the period 2017/10/02 - 2017/10/09.

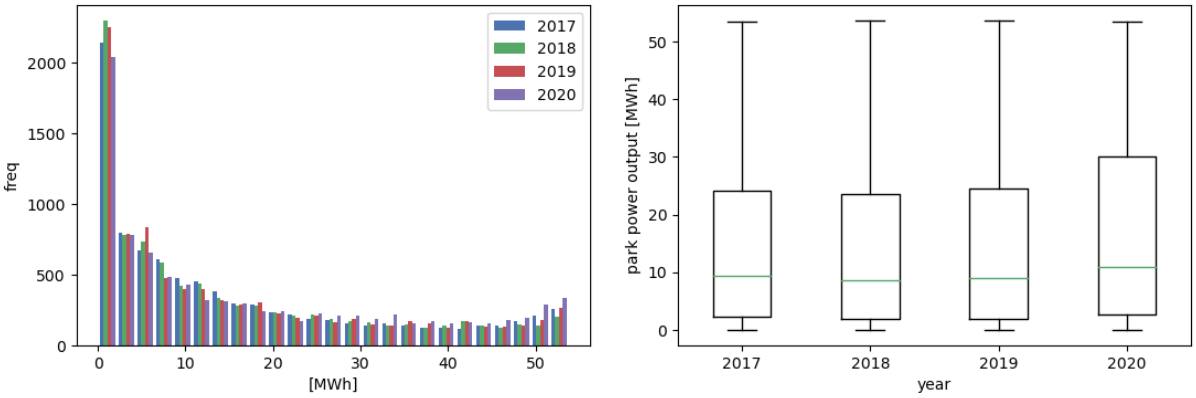
12.4 Comparing MEPS forecasts

Fig. 17 shows the comparison between the two modeled wind speeds with the on-site measurement for a week in October 2017. The figure does show, as commented in sec. 12.3 that under-predicting is not uncommon with the MEPS forecasts. Otherwise, the figures show that the two forecasts generally follow each other and there is no reason to conclude that the intraday weather forecast are substantially better than the day-ahead weather forecast.

Tab. 3 further compares the modeled weather features with the on site measurements for the whole year of 2017 using the normalized root mean square error (NRMSE) and linear correlation. The NRMSE error for the day-ahead data is higher than the intraday data by 1.3% and 0.3% respectively for wind speed and temperature. The correlation coefficient shows similarly small improvements for the intraday data.

Day-ahead/ Intraday	NRMSE	correlation coefficient
ws	0.115/0.108	0.695/0.739
wd	0.282/0.282	0.473/0.476
T	0.048/0.045	0.967/0.973

Table 3: NRMSE error and linear correlation between measured and modeled wind.



(a) Data distribution per year as histograms.

(b) Yearly variability.

Figure 18: Yearly data distribution and variability for park power output

12.5 Determining dataset split

Early testing of the models used the years 2017 and 2018 for training, 2019 for validation and 2020 for testing. These tests of the models showed discrepancies between validation and test loss over all the models, which implied that the problem lied with the datasets rather than being a overfitting problem. It was therefore necessary to investigate the properties seen in each year of the data, to determine if the problem was related to the dataset splitting.

Fig. 18 shows the data distributions and variability for the target variable for the years 2017-2020. Figure (a) shows that each year has a reasonably similar data distribution, which is important when determining dataset splitting between training, validation and testing. Figure (b), however, shows the box-plots for each year where the third quantile of year 2020 is somewhat higher than for the rest of the years. This increased variance can introduce a less than optimal model if 2020 is used as test data, since the model has trained on data with lower variance.

Considering that there exist some yearly variations as shown in fig. 14, a whole year should be set aside for testing, meaning that 25% of the data is used for testing. Therefore, the year 2019 will be used for testing while the years 2017,2018 and 2020 are reserved for training and validation. For the train/val split an approximation of the 80/20 split is chosen. Half a year is reserved for validation, which equals about 16.7% of the three years and leaves 83.3% for testing.

13 Neural Network-based Models

The neural network-based models are implemented using Pytorch Forecasting (Beitner, 2020), a high-level API specialized in the field of neural time series forecasting. It builds on the popular Python-based libraries Pytorch and Pytorch-lightning. The neural network models introduced in part. III are both implemented in Pytorch-Forecasting. For a practitioner, there are several advantages to using existing deep learning architectures implemented in a popular software library rather than implementing a new architecture:

- Developing a deep learning architecture from scratch can be a time-consuming process. By leveraging existing architectures implemented in libraries such as Pytorch Forecasting, it is possible to save a significant amount of time and effort. These architectures have already been designed, implemented, and optimized by experts, allowing the practitioner to focus more on the specific problem rather than reinventing the wheel.

- Established deep learning libraries often provide highly optimized implementations of popular architectures. These implementations have been thoroughly tested and refined, ensuring high performance and reliability. By using a well-established architecture, the practitioner can benefit from the collective expertise and community contributions that have improved its performance over time.

- If one is conducting research or working on a project that requires comparing results with existing works, using established architectures can help ensure reproducibility. Well-known architectures have established benchmarks and reported results, making it easier to compare and validate the findings against previous studies.

- Many deep learning libraries provide pre-trained models for popular architectures. These models are trained on large datasets and can be fine-tuned or used as a starting point for various tasks. Leveraging pre-trained models can significantly speed up the development process, as they often capture general-purpose features that can be useful across different domains.

- Popular deep-learning libraries have vibrant communities with active forums, online tutorials, and extensive documentation. If one encounters challenges or has questions while using an existing architecture, one can rely on community support to find solutions. The availability of resources and code examples can also facilitate the learning process and help to understand best practices in deep learning.

Each model is fitted separately on the two datasets, meaning that there in practice are instantiated four neural network models where two and two have the same architecture. As depicted in Fig. 13, the dataset contains 3 h of (linear) memory. The intraday models will thus use the previous 4-time steps as encoder input while predicting for the next 6 hours. As mentioned in 9.1, the day-ahead models will not use previous history and be cold start models. This was, however, not possible for DeepAR since it is an autoregressive model that uses the previous value in its prediction as depicted in Fig. 11b. Hence, the day-ahead DeepAR model must have at least 1-time step as a lookback window. In the Pytorch-Forecasting setup that would mean including the true production one hour

before production time, which as mentioned in sec. 9.1, has not happened yet at the prediction time. Therefore, the DeepAR model setup must predict from 12:00 the day before, meaning that its forecasting horizon is 36. Even though DeepAR predicts for a larger forecasting horizon, its performance will only be evaluated on the day-ahead time frame meaning that only the performance on the last 24 time steps will be evaluated.

DeepAR is designed to only include covariates in the form of known future values, whereas TFT offers a more comprehensive integration of covariates. This study uses the MEPS NWP data which is considered as known future inputs as they are available before prediction time. Furthermore, on-site weather measurements of wind speed serving as time-varying unknown variables are available and can therefore be used in the TFT as input to the encoder along with the target variable time-varying unknown variables.

Both TFT and DeepAR produce probabilistic predictions in the form of quantile forecasts using the percentiles $q \in [0.02, 0.1, 0.25, 0.5, 0.75, 0.9, 0.98]$. TFT directly computes the quantiles using QR, while DeepAR extracts the quantiles using the cumulative density function (cdf) of the predicted pdf. This way, it is easy to compare the two models even though they use different methods to produce probabilistic output.

Temporal Fusion Transformer								
Dataset	**Network configuration**							
	d_{model}	d_{attn}	h_{LSTM}	N_{LSTM}	lr	p	a	c
Dayahead	128	2	122	1	0.01	0.23	0	0.3
Intraday	52	2	27	2	0.00504	0.294	0.00189	0.3
DeepAR								
Dataset	**Network configuration**							
	h_{LSTM}	N_{LSTM}	lr	p	a	c		
Dayahead	20	3	0.003266	0.576	0.0011	0.17227		
Intraday	22	1	0.009	0	0.0018	0.008		

Table 4: Hyperparameter configuration for the neural network models

13.1 Hyperparameter configuration

The models contain a considerable amount of inner parameters, termed hyperparameters, including learning rate, network depth, and various regularization parameters. Finding the optimal hyperparameters can be a large task as there in practice can exist thousands of combinations. Smart searching algorithms are therefore vital for the increased use of ML models in production. For this thesis, hyperparameter searches were done using the ASHA algorithm (L. Li et al., 2018), which supports parallelism and prunes bad runs early on using successive halving (Jamieson & Talwalkar, 2015) to reduce searching time. The reader is directed to the papers (L. Li et al., 2018; Jamieson & Talwalkar, 2015) for further information.

Hyperparameters common to both models are learning rate lr, dropout rate p, weight decay constant a, and gradient clipping norm c. Additionally, both models utilize an LSTM network which is parameterized by hidden vector size h_{LSTM}, and the number of LSTM layers N_{LSTM}. The TFT is further parameterized by model dimension d_{model} and the number of attention heads d_{attn}.

The models were lightly tuned after the ASHA (L. Li et al., 2018) algorithm found suitable combinations of hyperparameters. The hyperparameters used in the final models are presented in Tab. 4.

13.2 Regularization

Several regularization techniques are used, therein early stopping, dropout, gradient clipping, and weight decay. Values for dropout rate, gradient clipping, and weight decay are included in the hyperparameter search space as they are highly dependent on the other hyperparameters. Early stopping is implemented similarly for all models and stops the training if the validation loss has not improved within a small threshold in the last 50 epochs. Dropout in TFT is implemented in various parts of its architecture, therein at each fully connected layer present in the GRN's, GLU's and the Add&Norm gates. Furthermore, if the model uses multiple LSTM layers there is also a dropout between these. Similarly, dropout is only implemented between the LSTM layers of DeepAR if there is more than one LSTM layer.

SARIMAX							
Dataset	**Network configuration**						
	p	d	q	P	D	Q	m
Dayahead	4	1	0	2	1	0	24
Intraday	4	1	0	2	1	0	24

Table 5: SARIMAX model parameters for the two datasets

13.3 Network training

Both of the networks are optimized by the ranger optimizer (Wright & Demeure, 2021), a synergistic optimizer with Adam (Chilimbi et al., 2014) and AdamW (Loshchilov & Hutter, 2017) as core components. The Ranger algorithm unites several modifications of the Adam and Adamw algorithms, in all 8 ideas which were found to be orthogonal and compatible to the Adam optimizer. The reader is directed to (Wright & Demeure, 2021) for a thorough explanation of the algorithm along with the 8 additional components. Ranger was found to give more stable training with fewer oscillations in my experiments and generally converged in most of the hyperparameter searches while the Adam optimizer needed greater attention during tuning and failed more often than not. This, by my hypothesis, can be attributed to the erratic nature of wind power resulting in the need of a highly adaptive optimizer that is robust to large variations.

14 SARIMAX model

SARIMAX is the only statistical WPF model used in this study. Similarly to DeepAR, SARIMAX must have a 1-time step as a lookback window since it is an autoregressive model. Therefore, the forecasting horizon in the day-ahead framework must be 36.

It is implemented using pmdarima (Smith et al., 2017–), a statistical Python library. The parameters of the SARIMAX model were fitted using the auto-arima function which first determines the order of differencing using statistical tests such as Kwiatkowski–Phillips–Schmidt–Shin (Kwiatkowski, Phillips, Schmidt, & Shin, 1992),
Augmented Dickey-Fuller (Dickey & Fuller, 1979) or Phillips–Perron (Phillips & Perron, 1988). Thereafter the auto-arima function test different combinations of p and q and calculates the AIC criterion for each model. The model with the lowest AIC score is chosen and returned. The results of the auto-arima function is presented in Tab. 5. The parameters found using auto-arima concur with Fig. 13, where the ACF tailed off and the PACF cut off early, implying that it is an AR model. As the time series is non-stationary, a differencing term has been used to make it stationary. The seasonal parameters are set to $m = 24$, since it is hourly data. The parameters P,D,Q was chosen by the auto-arima function and implies a seasonal offset of $P = 2$ and seasonal difference of $D = 1$.

15 Evaluation Metrics

15.1 Continuous Ranked Probability Score

When choosing evaluation metrics for probabilistic predictions, it is important to both measure the sharpness and coverage as discussed in sec. 4.4. Additionally, in this study, it is important to choose metrics that can measure the performance of the forecasts from all the models since they use different ways to produce probabilistic forecasts. Common to both Bayesian inference and frequentist inference like quantile regression is that they can be evaluated using quantile-based metrics. A popular metric to evaluate probabilistic forecasts is the continuous ranked probability score (CRPS) which can be calculated using quantiles. (Gneiting & Ranjan, 2011) does a thorough proof and discussion of quantile weighted CRPS which will be used in this study.

Given a density forecast, f, of time series y. F denotes the cumulative distribution function (CDF) of f, where quantile forecasts, can be computed with $F^{-1}(q)$, given significance level q. The CRPS can then be defined as a scaled integral over the quantile loss (QL):

$$CRPS(y, F) = \int_0^1 2QL(y, F^{-1}(q), q)dq \tag{45}$$

A discrete approximation of the CRPS can be defined, using $\hat{y}$ as the quantile output of the network and M as the number of quantiles:

$$CRPS(y, \hat{y}) \approx \frac{2}{M} \sum_{q_i \in \mathcal{Q}} QL(y, \hat{y}, q_i) \tag{46}$$

The approximation above is true if the quantiles are equally spaced, while the experiments in this study use quantiles that are not. To accommodate for the unequal spacing, $\frac{1}{M}$ must be replaced with the difference $dq_i = q_i - q_{i-1}$ in the sum.

$$CRPS(y, \hat{y}) \approx 2 \sum_{q_i \in \mathcal{Q}} QL(y, \hat{y}, q_i)dq_i \tag{47}$$

Quantile weighted CRPS, here denoted $qCRPS(y, \hat{y})$, are constructed by applying weights $v(q) \in [0, 1]$ to the integrand/sum.

$$qCRPS(y, \hat{y}) = 2 \sum_{q_i \in \mathcal{Q}} v(q_i)QL(y, \hat{y}, q_i)dq_i \tag{48}$$

Evaluating the whole multi-step forecast will further involve averaging over the forecasting horizon.

$$qCRPS(y, \hat{y}) = \frac{2}{T_{max}} \sum_{T=1}^{T_{max}} \sum_{q_i \in \mathcal{Q}} v(q_i)QL(y_t, \hat{y}, q_i)dq_i \tag{49}$$

The choice of weighing function depends on if the evaluation should put the most emphasis on the outer or center quantiles, or weigh the whole distribution equally and in that case eq. 48 reduces to the ordinary CRPS. This study will use both tail weighing of the form $v(q) = (2q - 1)^2$ and ordinary CRPS $v(q) = 1$ to evaluate the forecast. The reason for avoiding center weighing is that the tails of the distribution should be a more reliable rule for the wind power trader to keep within. While the median prediction is important, the tails of the distribution is of higher importance as they give information about the uncertainty of the predictions.

15.2 p-Risk

Since the SARIMA model computes probabilistic forecasting as a PI, and obtaining several PIs would involve running the model several times which can result in an issue typically called *quantile crossing* (Chernozhukov, Fernandez-Val, & Galichon, n.d.), another performance metric is needed to compare the forecasts from the neural network models with the ones from SARIMA. For this, a special case of the CRPS, called p-Risk, can be used to evaluate a single percentile. The 50 and 80 percentile risks, called P50- and P90-risk were used to evaluate the models in the papers introducing TFT and DeepAR (Lim et al., 2020; Salinas et al., 2020), and is therefore a good choice to also be consistent with previous work. The q-Risk for one test sample is defined as,

$$p - Risk = \frac{2 \sum_{y_t \in \Omega} \sum_{T=1}^{T_{max}} QL(y_t, \hat{y}, q),}{T_{max}}. \tag{50}$$

15.3 Prediction Interval Coverage Probability

When using the percentiles $q \in [0.02, 0.1, 0.25, 0.5, 0.75, 0.9, 0.98]$, three PI's are essentially constructed, $[0.02, 0.98], [0.1, 0.90], [0.25, 0.75]$ with respective significance levels $\alpha = [0.04, 0.20, 0.5]$. The PI's coverage should therefore be $0.96, 0.80, 0.50$ to be valid confidence intervals. This can be scored by using the prediction interval coverage probability (PICP), which measures the percentage of coverage for each of the PI's (Alcántara, Galván, & Aler, 2022)

$$PICP = \frac{1}{n_t} \sum_{i=1}^{n_t} c_i, \quad c_i = \begin{cases} 1 & y_t \in [L_i, L_u] \\ 0 & y_t \notin [L_i, L_u] \end{cases} \tag{51}$$

The optimal outcome is to have valid PIs, such that $\text{PICP}_\alpha \geq 1 - \alpha$, while keeping the width of the PI as small as possible.

CRPS / qCRPS	**TFT**	**DeepAR**
Day-ahead	**0.108** / **0.026**	0.121 / 0.028
Intraday	0.262 / 0.064	**0.078** / **0.018**

Table 6: CRPS and quantile weighted CRPS on the day-ahead and intraday datasets for the neural network models. (Lower (q)CRPS, the better)

P50 / P90-Risk	**TFT**	**DeepAR**	**SARIMAX**
Day-ahead	**0.153** / **0.081**	0.175 / 0.091	0.415 / 0.238
Intraday	0.367 / 0.154	**0.111** / **0.056**	0.184 / 0.099

Table 7: P50 and P90-Risk on the day-ahead and intraday datasets for all models. (Lower the p-Risk, the better)

16 Experimental Results

This section presents and discusses the results obtained from the experiments described in the previous sections.

16.1 Performance on Individual Datasets

Tab. 6 and Tab. 7 present the performance of the models on the day-ahead and intraday datasets using CRPS, tail weighted CRPS (qCRPS), and P50, P90-Risk. All of the evaluation metrics are negatively oriented, meaning that the best model is the one with the lowest score.

The day-ahead dataset
The tables show that TFT performs best in the day-ahead framework. The CRPS value implies that TFT has both the best and produces the most accurate predictions, and the tail-weighted CRPS implies that it also produces the most accurate tails. However, the difference between TFT and DeepAR for the day-ahead framework decreases from the CRPS to the qCRPS, implying that the TFT produces more accurate median predictions, while the two models can be expected to perform similarly well at the tails. Tab. 7 further show that TFT performs better than DeepAR on both the 50th and 90th percentile, also here is the difference less for the P90 risk. Both neural network models perform noticeably better than SARIMAX on the day-ahead data, not surprising as ARIMA models are known to perform poorly when the forecasting horizon increases.

The intraday dataset
With the intraday dataset, DeepAR outperforms the other models, while TFT performs inadequately. It is clear that the two autoregressive models perform best when the forecasting horizon is short, while TFT is designed to model longer forecasting horizons. Both DeepAR and SARIMAX's performance improves from the day-ahead forecasting model to the corresponding intraday model. This is expected since there still exists some memory from the lookback window, which introduces the question as to why TFT performs so much worse on the intraday dataset.

16.2 Performance Discussion

Figs. 19, 21, show the aggregated predictions of the three models for the days 2019/04/30 to 2019/05/06 respectively for the day-ahead and the intraday framework. Similarly, Figs. 20, 22, show the aggregated predictions for the days 2019/03/26 to 2019/04/02. All of the figures show the corresponding measured, and modeled wind speeds for the same time periods. The date ranges were chosen because they together provided a good representation of the wind speeds exhibited at Fakken wind park. With these visualizations, it is possible to further discuss the performance of the different models and compare the predictions with the measured and modeled wind speeds.

16.2.1 Network type

Temporal Fusion Transformer
TFT had a more extensive incorporation of covariates in its models. Both by including covariates in different forms, but also by including variable selection networks such that the influence of each covariate time step is weighted. This is reflected in the day-ahead predictions seen in the figures 19a, 20a, where the point predictions very much follow the modeled wind speed, especially in Fig. 20a. Most of the weakest predictions can be explained by discrepancies between the measured and modeled wind speeds, while the others show that TFT regularly underpredicts rather than overpredicts. My theory on this is that it puts too much weight on the modeled wind speed, and has not properly learned the cubic relationship between wind speed and power output. This might be because of the large spread seen in the intermediate wind speeds, 7-12 m/s, seen in Fig. 16a. Having a large spread means that these wind speeds are difficult to model.

The intraday predictions from TFT in figures 21a, 22a, are outright poor and it is clear that the model is extremely underfitted. The model was difficult to train, as few hyperparameter combinations came out with good loss surfaces. This implies that the intraday framework was perhaps too simple and incompatible with the TFT. The Attention mechanism processes the input as one set, meaning that using too few time steps might not be enough for the model to 'see' and model the whole day. In hindsight, choosing a longer forecasting horizon would be beneficial for the TFT. It could then use the lookback window, meaning that it might manage to better model the cubic relationship which was its problems with the day-ahead framework.

DeepAR
DeepAR performs much better on the intraday dataset than on the day-ahead dataset, which is already attributed to its autoregressive nature. The day-ahead predictions in figures 19b, 20b, are very conservative, and it seems to be using more weight on the previous predictions than the TFT did. Fig. 19b is a good example of where DeepAR is more conservative than the TFT, resulting in the TFT having sharper turns/corners (in lack of a better word). When inspecting the shape of the point predictions of DeepAR, they almost perfectly resemble the shape of the modeled wind speed. Similarly, as with the TFT, DeepAR struggles to learn the cubic relationship between wind speed and wind power. Additionally, it seems to also struggle with learning the cut-in wind speed.

PICP $1-\alpha$			
$\alpha = 0.04$	**TFT**	**DeepAR**	**SARIMAX**
Day-ahead	0.893	**0.958**	NA
Intraday	0.547	**0.946**	NA
$\alpha = 0.20$			
Day-ahead	0.736	**0.792**	0.940
Intraday	0.359	**0.821**	0.859
$\alpha = 0.5$			
Day-ahead	0.466	**0.484**	NA
Intraday	0.192	**0.544**	NA

Table 8: Prediction interval coverage probability for significance levels $\alpha = [0.04, 0.2, 0.5]$. The most valid PI's are highlighted in bold.

DeepAR achieves good performance when predicting for the intraday market, as the predictions improve from the day-ahead predictions and it shows that it utilizes the lookback window in its predictions seen in figures 21b, 22b. The conservativeness seen in the day-ahead predictions is no longer visible in the intraday predictions and DeepAR maps the modeled wind speed very well. The weakest predictions can be explained by discrepancies between the measured and modeled wind speeds, similar to the day-ahead predictions of TFT. The combination of day-ahead predictions from TFT and intraday predictions from DeepAR seems to be a good fit, seeing as what is lacking in the day-ahead forecasts is mostly present in the intraday forecasts.

SARIMAX

Figs. 19c, 20c, show very different qualities of prediction. Where the predictions in the first figure mostly follow the true values, while the predictions in the last figure are significantly inaccurate. These experiments show that SARIMAX is not a robust forecasting method for the day-ahead framework, where both the figures and the P50-, P90-Risk reflect this.

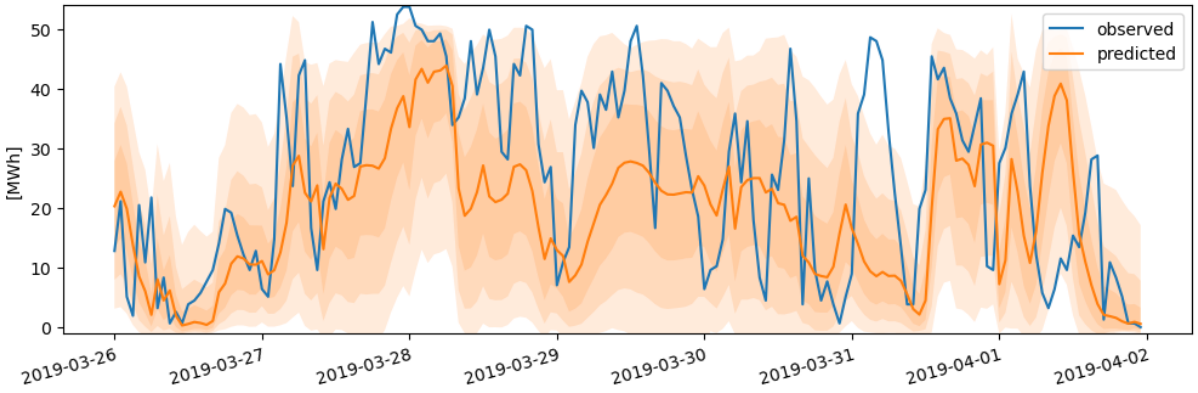
SARIMAX produces satisfactory intraday predictions, as seen in figures 21c, 22c. The point predictions are better than TFT's, and less conservative than DeepAR, proving that for some problems, it is not really necessary with deep neural networks. However, SARIMAX struggles with keeping inside the production limits $[0, 54]$ and regularly predicts both negative production and above the maximum possible production. The intraday predictions are also lagged, similar to the day-ahead predictions.

16.2.2 Prediction Intervals

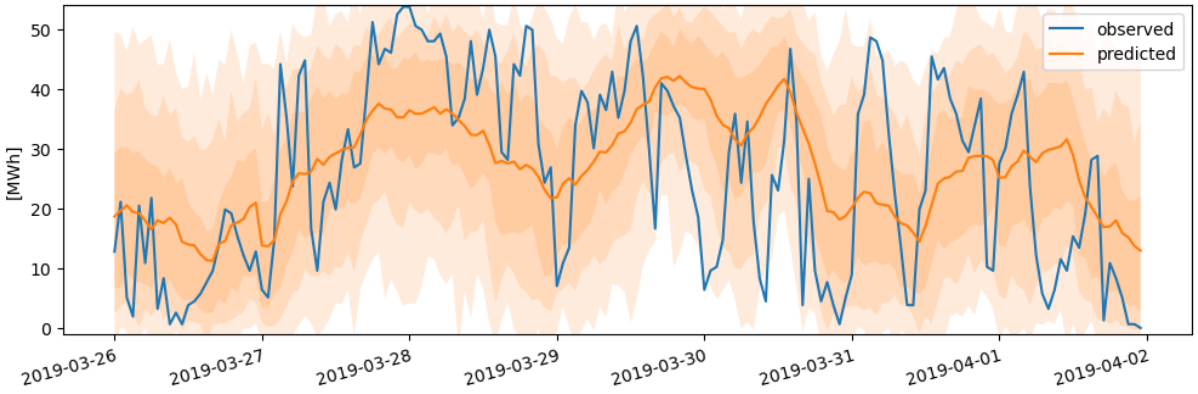
Tab. 8 presents the average coverage of the PI. The table shows that DeepAR produces the most valid PIs, both for the day-ahead and the intraday framework. The day-ahead TFT lies about 4-7 % below for each of the PIs, which is reflected in Figs. 19a, 20a, where it can be seen that DeepAR produces wider PIs. This is a common trade-off as correct coverage often results in increased width, which can reduce the interpretability of the prediction. One could argue that the predicted distribution of the TFT reflects the variability of the data better. The PI's of the day-ahead DeepAR seem to have a nearly constant width, while the day-ahead TFT has a more varying width. Especially the low wind speeds give a low uncertainty with the TFT, which actually reflects the spread seen in Fig. 16a. In contrast, DeepAR is almost equally uncertain about all of the wind speed ranges. Obviously, this can be partially resolved by using PI with lower coverage.

Not surprisingly, the intraday TFT does not provide the specified coverage at all. The intraday DeepAR, however, produces valid PIs where 0.8 and 0.5 actually exceed their specified coverage. Despite having good coverage, the PIs are remarkably acute and no longer show the non-varying width. Overall, the intraday DeepAR performs very well given the input it is given.

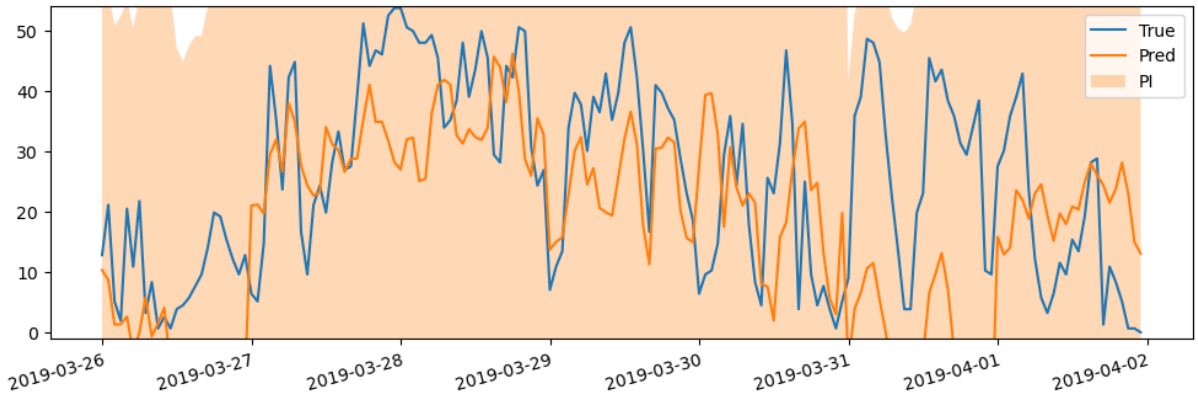
Figs. 19c, 20c show that SARIMAX produces extremely wide PIs for the day-ahead framework while producing significantly sharper PI for the intraday framework. This was, however, not unexpected as a forecasting horizon of 36 is far too long for an ARIMA model. Similarly to the point predictions, the coverage and width produced by the intraday model are satisfactory.



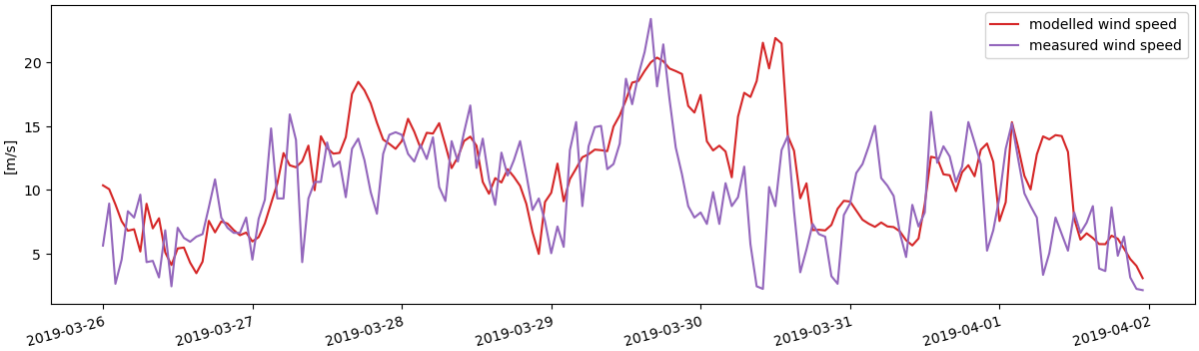
(a) Visualization of predictions from TFT



(b) Visualization of predictions from DeepAR

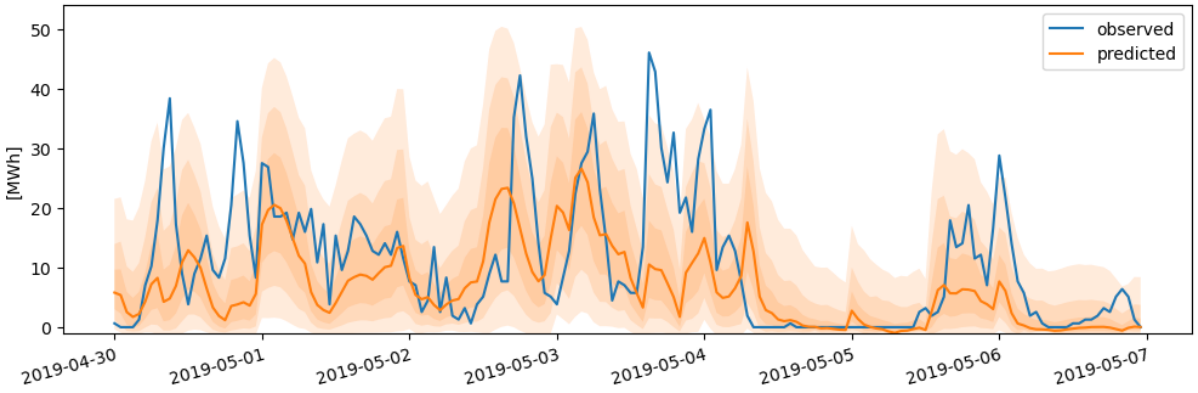


(c) Visualization of predictions from SARIMA

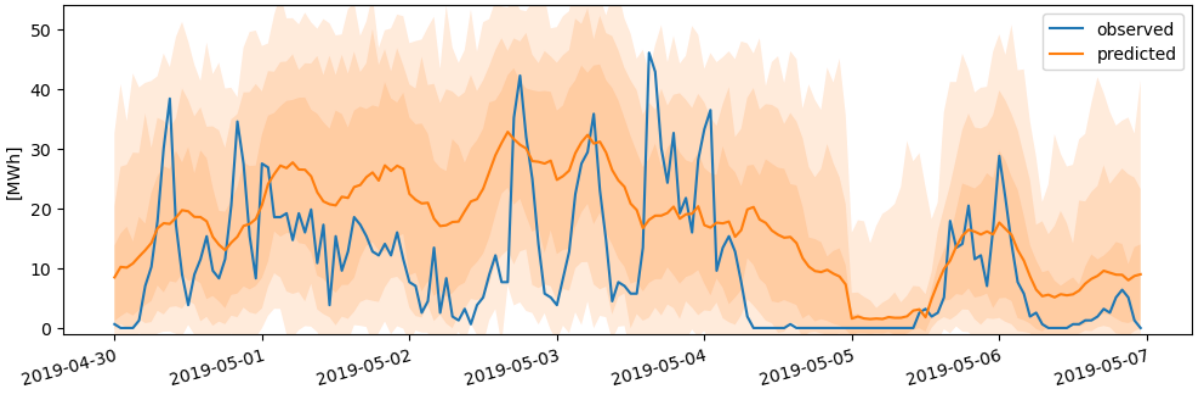


(d) Modeled and measured wind speed

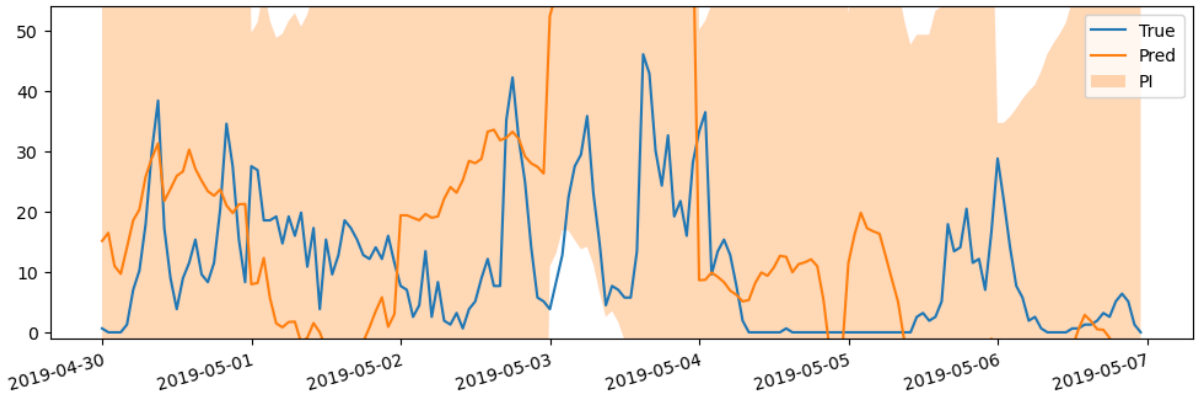
Figure 19: Aggregated predicted park power output for the day-ahead market, along with modeled and measured wind speed, for the week 2019/03/26-2019/04/02



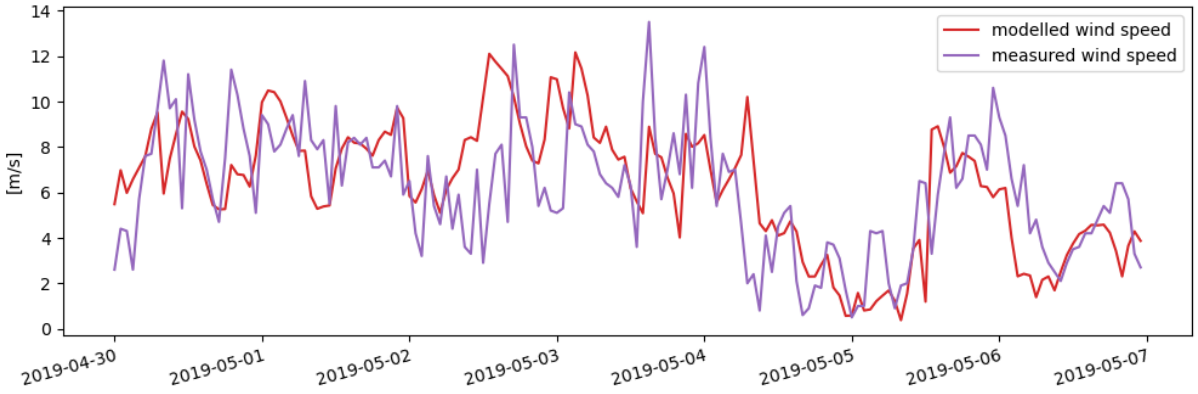
(a) Visualization of predictions from TFT



(b) Visualization of predictions from DeepAR

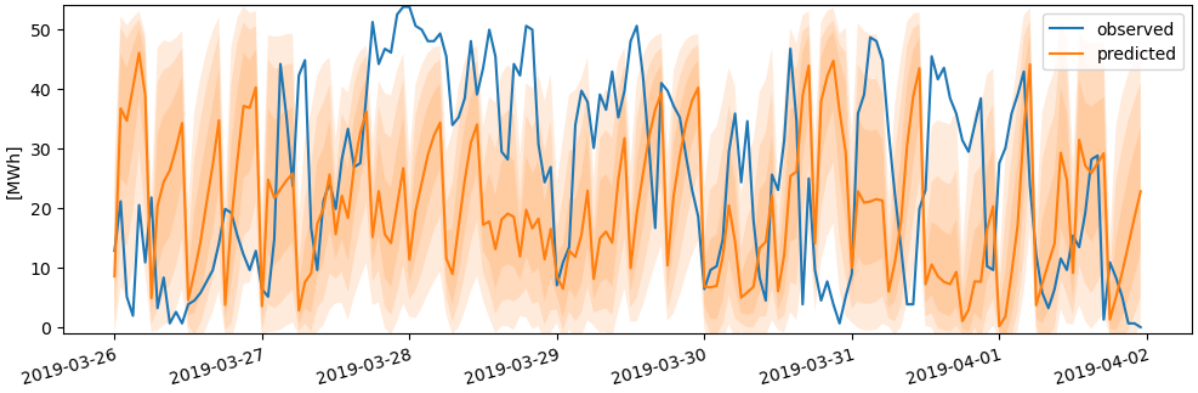


(c) Visualization of predictions from SARIMA

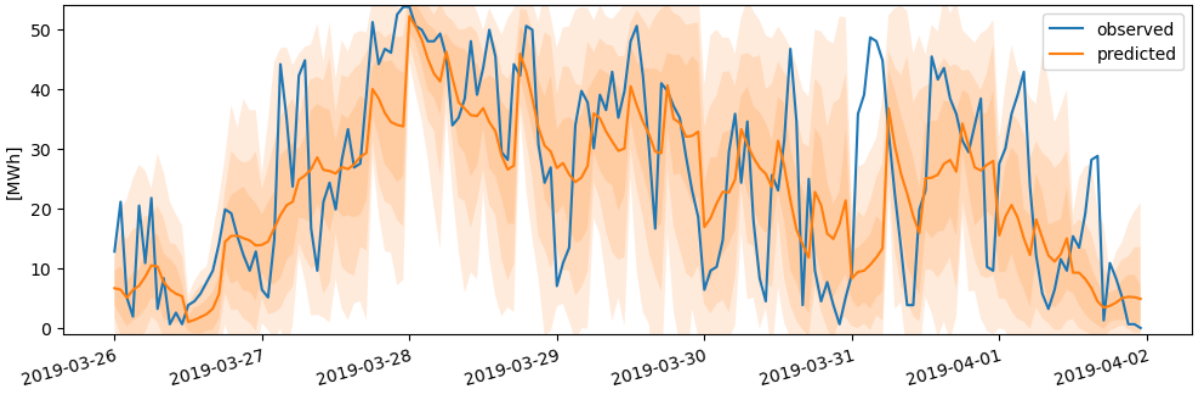


(d) Modeled and measured wind speed

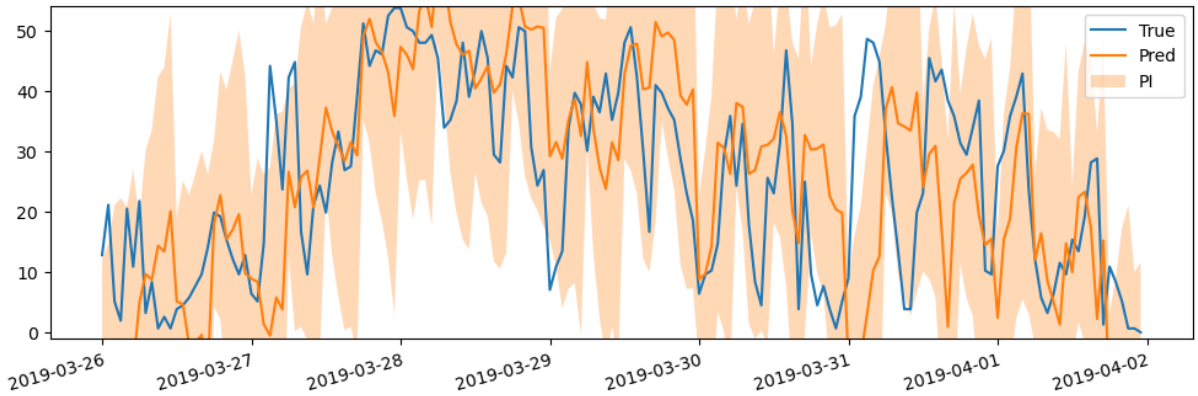
Figure 20: Aggregated predicted park power output for the day-ahead market, along with modeled and measured wind speed, for the week 2019/04/30-2019/05/06



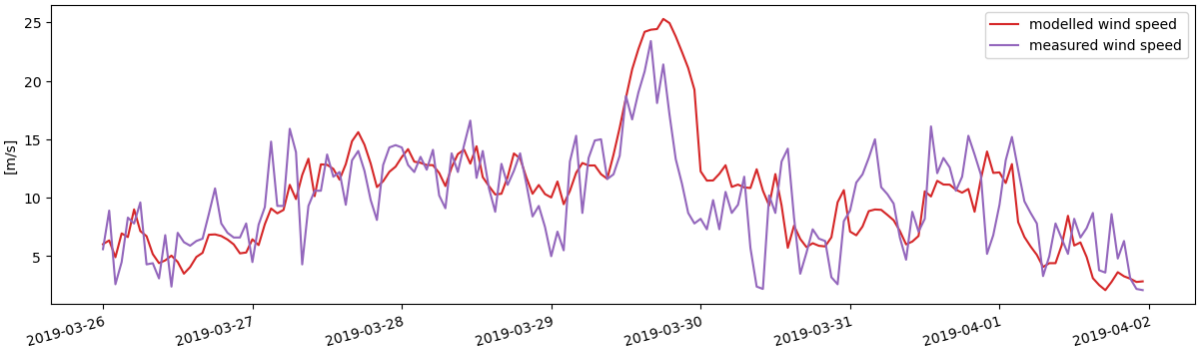
(a) Visualization of predictions from TFT



(b) Visualization of predictions from DeepAR

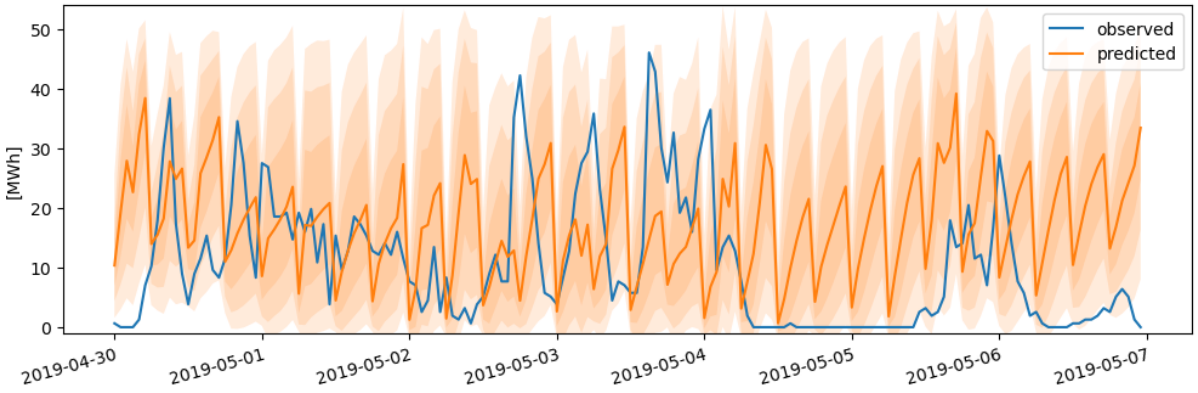


(c) Visualization of predictions from SARIMA

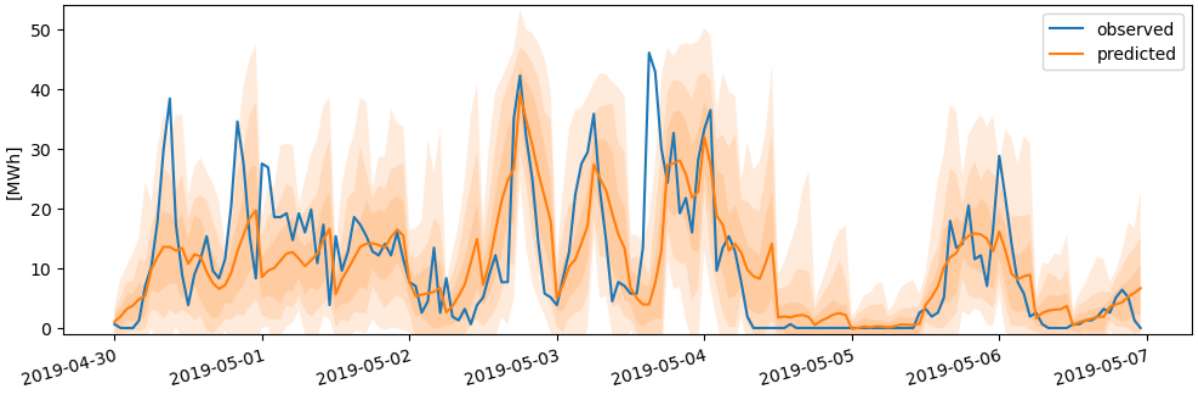


(d) Modeled and measured wind speed

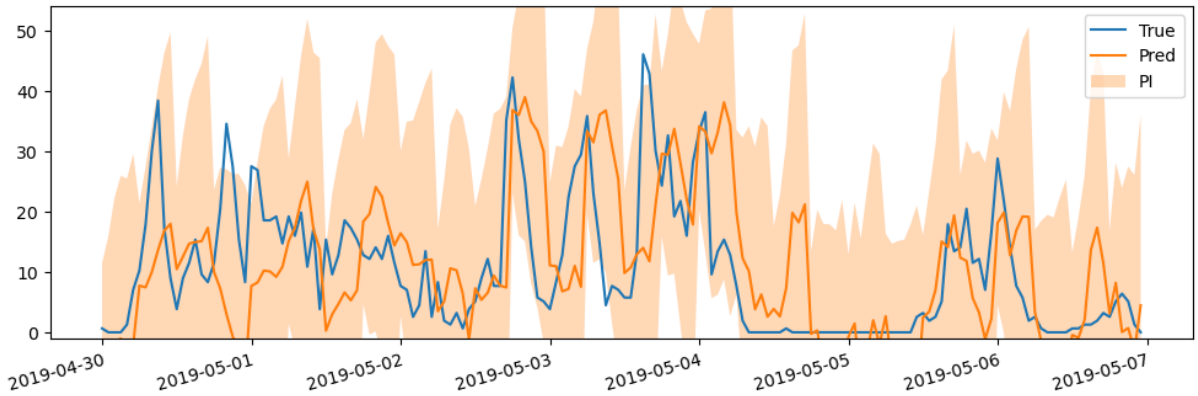
Figure 21: Aggregated predicted park power output for the intraday market, along with modeled and measured wind speed, for the week 2019/03/26-2019/04/02



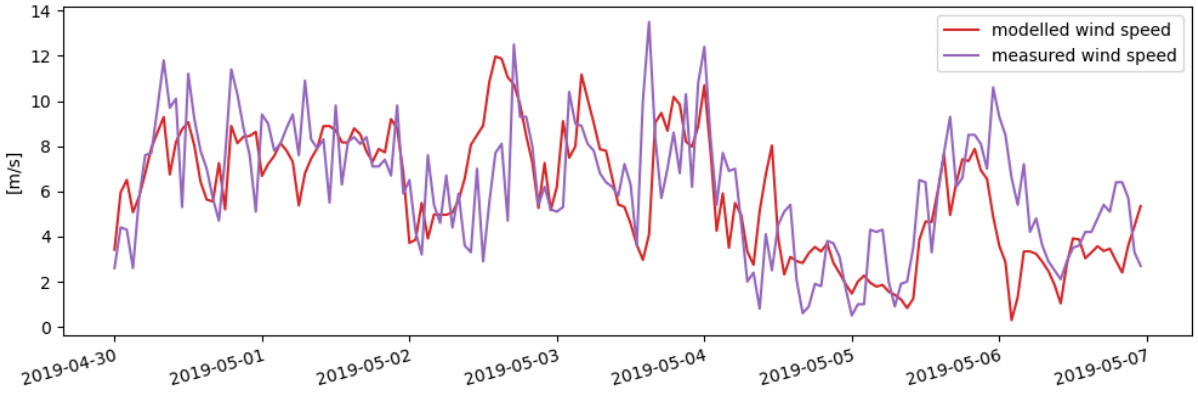
(a) Visualization of predictions from TFT



(b) Visualization of predictions from DeepAR



(c) Visualization of predictions from SARIMA



(d) Modeled and measured wind speed

Figure 22: Aggregated predicted park power output for the intraday market, along with modeled and measured wind speed, for the week 2019/04/30-2019/05/06

Part V / Conclusions

This thesis studied the problem of probabilistic wind power forecasting at Fakken wind park, focusing on producing predictions that encapsulate the uncertainties seen in wind power. This chapter is organized into three sections. First, the concluding remarks will be presented, where the key findings and contributions of this study are summarized. Secondly, my experience with the forecasting library Pytorch-Forecasting will be summarized. Lastly, potential further research for wind power forecasting at Fakken wind park will be discussed.

17 Concluding remarks

The following key findings and conclusions are found in this study:

- Transformer models are indicated to be good alternatives for the day-ahead framework due to the good performance of the Temporal Fusion Transformer (TFT). However, further investigation with several Transformer architectures should be explored to correctly asses their performance in day-ahead wind power forecasting.
- When deciding between the TFT and DeepAR for the day-ahead framework, the risk-awareness of the power producer must be taken into account. While TFT exceeds in accuracy, DeepAR offers a more comprehensive assessment of risk, making it a suitable model for those who prioritize risk-awareness in their bidding strategies.
- Autoregressive models have shown great promise for the intraday framework, where the DeepAR provided both accurate and properly risk aware predictions. Therefore, autoregressive models seems to be a good initial choice for short-term wind power forecasting.
- SARIMAX, although not inherently a poor choice for intraday forecasting, will require further time and effort in model creation and parameter tuning to achieve its optimal performance. While auto-arima provides initial guidance for parameter selection, the practitioner should manually create and train the SARIMAX model for best results.
- When deciding between DeepAR and SARIMAX for the intraday framework, a trade-off between accuracy and model complexity must be taken into account. DeepAR offers higher accuracy at the expense of model complexity, while SARIMAX provides a more interpretable approach.
- Wind power forecasting models are highly dependent on the performance of the numerical weather predictions used as exogenous variables. Therefore, the performance of wind power forecasting models will only improve further as numerical weather predictions improve.

The goal of these findings is to provide insights for wind power traders and forecasters, enabling them to make informed decisions when selecting forecasting models based on their specific requirements. Overall, the research presented in this study opens courses for further exploration and improvements in wind power forecasting at Fakken wind park.

18 Pytorch Forecasting

This section will comment on my experience in using the forecasting library Pytorch-Forecasting (Beitner, 2020). Overall, the models used from Pytorch-Forecasting, TFT, and DeepAR, worked well when used as described in the tutorials from Pytorch-Forecasting. However, working with Pytorch-Forecasting gave limited flexibility when trying to do something outside of the defined features.

One feature that I wanted to fix, but was not possible to fix was how it passed data to the models. The method for passing data was a sliding window as described in sec. 4.3, however, it was not possible to change the step size of the window and it was set to one. Meaning that window moved only one step at a time, whereas for the day-ahead and intraday framework the step size should have been 24 and 6. To accommodate for this, the predictions had to be filtered, such that only the correct prediction times were chosen and scored. The fact that the step size is a constant of one is surprising as it is not difficult to implement, and it is very common for time series forecasting to use sliding windows with a step size equal to the window size. Another feature that was possible was to create yourself was to implement your own metrics using their framework. Nonetheless, I was not able to create personal metrics due to its complexity and poor documentation. In the end, all of the correct predictions were stored and the metrics were implemented outside of the Pytorch-Forecasting framework.

Even though using the library comes with limited flexibility, this is something that most software engineers will find frustrating. Practitioners with less software experience will probably find it comforting that it just works when using it for the standard tasks presented in the tutorial. However, whenever a problem occurs one needs a technical background to be able to sort the issue out. Additionally, the output of the networks is not intuitive and also there it helped with a technical background to figure out how to properly use the outputs.

In conclusion, Pytorch-Forecasting is a good alternative if e.g., a company has a restrained software team, or if researchers want to easily implement some of its models as baselines. It is a good first step to check if deep learning might be suitable for the problem at hand. However, it's important to note that there may be scenarios where designing a new architecture is necessary or beneficial. For specialized tasks or domains with unique requirements, a custom architecture might be more suitable. Additionally, if you have expertise in deep learning and want to explore innovative ideas or improve upon existing architectures, designing your own model can be an exciting and rewarding endeavor.

19 Further work

Based on the results of this study, the following further work is advised. With the aim of producing results with higher accuracy, the first step is to revisit the data preprocessing methods. Specifically, missing values imputation and data splitting can be revisited. Missing values imputation was in this study done by padding the missing values by the last valid value, which resulted in regions as large as a day being padded with only one value. More advanced methods such as the maximum likelihood imputation proposed in (García, Luengo, & Herrera, 2015) or spatio-temporal imputation techniques (Cini, Marisca, & Alippi, 2022) could be considered to improve the prediction performance.

Additionally, only two data splitting configurations were explored, while other configurations and even other evaluation methods like cross-validation for time series (Bergmeir, Hyndman, & Koo, 2018) can be explored. Additionally, this study did not investigate outlier and anomaly detection, which can significantly reduce noise in the dataset. For wind power data, bottom-curve stacked anomalies as described in (Wang, Hu, Li, Foley, & Srinivasan, 2019) can be removed and imputed using the chosen missing value imputation method. Furthermore, statistical tests can be used to detect possible outliers as described in (Ben-Gal, 2005).

As discussed in sec. 16.2, the models struggled with learning the cubic relationship with wind speed and did not fully learn the cut-in, rated, and cut-off wind speeds. Therefore, new feature engineering specifically targeting these issues can be explored. I would recommend trying two features. Firstly, a feature with the polynomial of modeled wind speed can potentially help map the non-linear relationship between wind speed and wind power. I would recommend to start with the order of three, since this appears in the wind power equation 19. Secondly, a categorical feature that categorizes the operational regions based on modeled wind speed, e.g., below cut-in, in between cut-in and rated, rated and cut-off, and above cut-off wind speed.

Due to the limitations of working with Pytorch-Forecasting, another recommendation is to move forward with the insights from this study and create models using other deep learning libraries, e.g., the ones that Pytorch-Forecasting builds upon. A good starting point is to continue with using Attention-based models for the longer time frames used in day-ahead forecasting, and LSTM-based models for the shorter time frames used in intraday forecasting. Another suggestion is to use either the day-ahead TFT or DeepAR and the intraday DeepAR model from this study in an economic evaluation seeking to increase the income of the wind power producer as done in (Mazzi & Pinson, 2017).

References

Alcántara, A., Galván, I. M., & Aler, R. (2022). Direct estimation of prediction intervals for solar and wind regional energy forecasting with deep neural networks. *Engineering Applications of Artificial Intelligence*, *114*, 105128. Retrieved from https://www.sciencedirect.com/science/article/pii/S0952197622002573 doi: https://doi.org/10.1016/j.engappai.2022.105128

Bai, S., Kolter, J. Z., & Koltun, V. (2018). *An empirical evaluation of generic convolutional and recurrent networks for sequence modeling.* arXiv. Retrieved from https://arxiv.org/abs/1803.01271 doi: 10.48550/ARXIV.1803.01271

Beitner, J. (2020). *Pytorch forecasting.* https://pytorch-forecasting.readthedocs.io/en/stable/index.html.

Ben-Gal, I. (2005). Outlier detection. In O. Maimon & L. Rokach (Eds.), *Data mining and knowledge discovery handbook* (pp. 131–146). Boston, MA: Springer US. Retrieved from https://doi.org/10.1007/0-387-25465-X_7 doi: 10.1007/0-387-25465-X_7

Bergmeir, C., Hyndman, R. J., & Koo, B. (2018). A note on the validity of cross-validation for evaluating autoregressive time series prediction. *Computational Statistics Data Analysis*, *120*, 70-83. Retrieved from https://www.sciencedirect.com/science/article/pii/S0167947317302384 doi: https://doi.org/10.1016/j.csda.2017.11.003

Bianchi, F. M., De Santis, E., Rizzi, A., & Sadeghian, A. (2015). Short-term electric load forecasting using echo state networks and pca decomposition. *IEEE Access*, *3*, 1931-1943. doi: 10.1109/ACCESS.2015.2485943

Bianchi, F. M., Livi, L., & Alippi, C. (2016). Investigating echo-state networks dynamics by means of recurrence analysis. *IEEE transactions on neural networks and learning systems*, *29*(2), 427–439.

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

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

Bianchi, F. M., Scardapane, S., Løkse, S., & Jenssen, R. (2020). Reservoir computing approaches for representation and classification of multivariate time series. *IEEE transactions on neural networks and learning systems*, *32*(5), 2169–2179.

Bianchi, F. M., Scardapane, S., Uncini, A., Rizzi, A., & Sadeghian, A. (2015). Prediction of telephone calls load using echo state network with exogenous variables. *Neural Networks*, *71*, 204-213. doi: https://doi.org/10.1016/j.neunet.2015.08.010

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

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

Brownlee, J. (2019). *How to use learning curves to diagnose machine learning model performance.* https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/.

Chernozhukov, V., Fernandez-Val, I., & Galichon, A. (n.d.). Retrieved from https://doi.org/10.3982 doi: 10.3982/ecta7880

Chilimbi, T., Suzue, Y., Apacible, J., & Kalyanaraman, K. (2014). Project adam: Building an efficient and scalable deep learning training system. In *11th {USENIX} symposium on operating systems design and implementation ({OSDI} 14)* (pp. 571–582).

Cho, K., van Merrienboer, B., Bahdanau, D., & Bengio, Y. (2014). *On the properties of neural machine translation: Encoder-decoder approaches.* arXiv. Retrieved from https://arxiv.org/abs/1409.1259 doi: 10.48550/ARXIV.1409.1259

Choi, R. Y., Coyner, A. S., Kalpathy-Cramer, J., Chiang, M. F., & Campbell, J. P. (2020, 02). Introduction to Machine Learning, Neural Networks, and Deep Learning. *Translational Vision Science Technology*, *9*(2), 14-14. Retrieved from https://doi.org/10.1167/tvst.9.2.14 doi: 10.1167/tvst.9.2.14

Cini, A., Marisca, I., & Alippi, C. (2022). Filling the g_ap_s: Multivariate time series imputation by graph neural networks. In *International conference on learning representations.* Retrieved from https://openreview.net/forum?id=kOu3-S3wJ7

Cini, A., Marisca, I., Bianchi, F. M., & Alippi, C. (2023). Scalable spatiotemporal graph neural networks. *Proceedings of the 37th AAAI Conference on Artificial Intelligence*.

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

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

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

Eikeland, O. F., Hovem, F. D., Olsen, T. E., Chiesa, M., & Bianchi, F. M. (2022). *Probabilistic forecasts of wind power generation in regions with complex topography using deep learning methods: An arctic case.* arXiv. Retrieved from https://arxiv.org/abs/2203.07080 doi: 10.48550/ARXIV.2203.07080

Fornacon-Wood, I., Mistry, H., Johnson-Hart, C., Faivre-Finn, C., P.B. O'Connor, J., & J. Price, G. (2022). Understanding the differences between bayesian and frequentist statistics. *International Journal of Radiation Oncolgy, Biology and Physics*, *112*, 1076-1082. doi: https://doi.org/10.1016/j.ijrobp.2021.12.011

Frogner, I.-L., Singleton, A. T., Kø ltzow, M. O., & Andrae, U. (2019). Convection-permitting ensembles: Challenges related to their design and use. *Quarterly Journal of the Royal Meteorological Society*, *145*(S1), 90-106. doi: https://doi.org/10.1002/qj.3525

García, S., Luengo, J., & Herrera, F. (2015). *Data preprocessing in data mining* (Vol. 72). Springer.

García, S., Luengo, J., & Herrera, F. (2015). *Data preprocessing in data mining* (1st ed.). Springer Cham. Retrieved from https://doi.org/10.1007/978-3-319-10247-4 (Published: 11 September 2014 (hardcover), 10 September 2016 (softcover), 30 August 2014 (eBook)) doi: 10.1007/978-3-319-10247-4

Gasthaus, J., Benidis, K., Wang, Y., Rangapuram, S. S., Salinas, D., Flunkert, V., & Januschowski, T. (2019). Probabilistic forecasting with spline quantile function rnns. In K. Chaudhuri & M. Sugiyama (Eds.), *The 22nd international conference on*

artificial intelligence and statistics, aistats 2019, 16-18 april 2019, naha, okinawa, japan (Vol. 89, p. 1901-1910). PMLR. Retrieved from http://proceedings.mlr.press/v89/gasthaus19a.html

Giebel, G., & Kariniotakis, G. (2017). 3 - wind power forecasting—a review of the state of the art. In G. Kariniotakis (Ed.), *Renewable energy forecasting* (p. 59-109). Woodhead Publishing. Retrieved from https://www.sciencedirect.com/science/article/pii/B9780081005040000032 doi: https://doi.org/10.1016/B978-0-08-100504-0.00003-2

Glorot, X., & Bengio, Y. (2010, 13–15 May). Understanding the difficulty of training deep feedforward neural networks. In Y. W. Teh & M. Titterington (Eds.), *Proceedings of the thirteenth international conference on artificial intelligence and statistics* (Vol. 9, pp. 249–256). Chia Laguna Resort, Sardinia, Italy: PMLR. Retrieved from https://proceedings.mlr.press/v9/glorot10a.html

Gneiting, T. (2011). Making and evaluating point forecasts. *Journal of the American Statistical Association*, *106*(494), 746-762. Retrieved from https://doi.org/10.1198/jasa.2011.r10138 doi: 10.1198/jasa.2011.r10138

Gneiting, T., & Ranjan, R. (2011). Comparing density forecasts using threshold- and quantile-weighted scoring rules. *Journal of Business & Economic Statistics*, *29*(3), 411–422. doi: 10.1198/jbes.2010.08110

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

Greff, K., Srivastava, R. K., Koutník, J., Steunebrink, B. R., & Schmidhuber, J. (2016). Lstm: A search space odyssey. *IEEE transactions on neural networks and learning systems*, *28*(10), 2222–2232.

Gudivada, V., Apon, A., & Ding, J. (2017). Data quality considerations for big data and machine learning: Going beyond data cleaning and transformations. *International Journal on Advances in Software*, *10*(1), 1–20.

Hanifi, S., Liu, X., Lin, Z., & Lotfian, S. (2020, July). A Critical Review of Wind Power Forecasting Methods—Past, Present and Future. *Energies*, *13*(15), 1-24. Retrieved from https://ideas.repec.org/a/gam/jeners/v13y2020i15p3764-d387902.html

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

Hochreiter, S. (1998, 04). The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, *6*, 107-116. doi: 10.1142/S0218488598000094

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, *9*(8), 1735–1780.

Hyndman, R., Koehler, A., Ord, J., & Snyder, R. (2008). *Forecasting with exponential smoothing: The state space approach*. Springer Berlin Heidelberg. Retrieved from https://books.google.no/books?id=GSyzox8Lu9YC

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

Jacobsen, M. (2014). *Short-term wind power prediction based on markov chain and numerical weather prediction models: A case study of fakken wind farm* (Unpublished master's thesis). UiT The Arctic University of Norway.

Jamieson, K., & Talwalkar, A. (2015). *Non-stochastic best arm identification and hyper-*

parameter optimization.

Jenkins, G. M. (1970). *Time series analysis; forecasting and control [by] george ep box and gwilym m. jenkins*. San Francisco: Holden-Day.

Jensen, V., Bianchi, F. M., & Anfinsen, S. N. (2022). Ensemble conformalized quantile regression for probabilistic time series forecasting. *IEEE Transactions on Neural Networks and Learning Systems*.

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

Jozefowicz, R., Zaremba, W., & Sutskever, I. (2015). An empirical exploration of recurrent network architectures. In *Proceedings of the 32nd international conference on machine learning* (pp. 2342–2350). Retrieved from https://proceedings.mlr.press/v37/jozefowicz15.pdf doi: 10.1145/3045118.3045367

Kalchbrenner, N., & Blunsom, P. (2013). Recurrent continuous translation models. In *Proceedings of the 2013 conference on empirical methods in natural language processing* (pp. 1700–1709). Retrieved from https://aclanthology.org/D13-1176/ doi: 10.1162/WN.2014.36.2.127

Kirchgässner, G., Wolters, J., & Hassler, U. (2012). *Introduction to modern time series analysis*. Springer Berlin Heidelberg. Retrieved from https://books.google.no/books?id=AfBunhJtxqwC

Kwiatkowski, D., Phillips, P. C., Schmidt, P., & Shin, Y. (1992). Testing the null hypothesis of stationarity against the alternative of a unit root: How sure are we that economic time series have a unit root? *Journal of Econometrics*, *54*(1), 159-178. Retrieved from https://www.sciencedirect.com/science/article/pii/030440769290104Y doi: https://doi.org/10.1016/0304-4076(92)90104-Y

Li, H., Xu, Z., Taylor, G., & Goldstein, T. (2017). Visualizing the loss landscape of neural nets. *CoRR*, *abs/1712.09913*. Retrieved from http://arxiv.org/abs/1712.09913

Li, L., Jamieson, K. G., Rostamizadeh, A., Gonina, E., Hardt, M., Recht, B., & Talwalkar, A. (2018). Massively parallel hyperparameter tuning. *CoRR*, *abs/1810.05934*. Retrieved from http://arxiv.org/abs/1810.05934

Lim, B., Arik, S. O., Loeff, N., & Pfister, T. (2020). *Temporal fusion transformers for interpretable multi-horizon time series forecasting.*

Lim, B., & Zohren, S. (2021). Time-series forecasting with deep learning: a survey. *Philosophical Transactions of the Royal Society A*, *379*(2199), 20200209. doi: http://doi.org/10.1098/rsta.2020.0209

Loshchilov, I., & Hutter, F. (2017). Fixing weight decay regularization in adam. *CoRR*, *abs/1711.05101*. Retrieved from http://arxiv.org/abs/1711.05101

Manwell, J., McGowan, J., & Rogers, A. (2010). *Wind energy explained: Theory, design and application*. Wiley. Retrieved from https://books.google.no/books?id=roaTx_Of0vAC

Mashlakov, A., Kuronen, T., Lensu, L., Kaarna, A., & Honkapuro, S. (2021). Assessing the performance of deep learning models for multivariate probabilistic energy forecasting. *Applied Energy*, *285*, 116405. Retrieved from https://www.sciencedirect.com/science/article/pii/S0306261920317748 doi: https://doi.org/10.1016/j.apenergy.2020.116405

Mazzi, N., & Pinson, P. (2017). 10 - wind power in electricity markets and the value of forecasting. In G. Kariniotakis (Ed.), *Renewable energy forecasting* (p. 259-278). Woodhead Publishing. Retrieved from https://www.sciencedirect.com/

science/article/pii/B978008100504000010X doi: https://doi.org/10.1016/B978-0-08-100504-0.00010-X

Mehandzhiyski, V. (2023). *What is an arimax model?* https://365datascience.com/tutorials/python-tutorials/arimax/.

Mikalsen, K. Ø., Bianchi, F. M., Soguero-Ruiz, C., & Jenssen, R. (2018). Time series cluster kernel for learning similarities between multivariate time series with missing data. *Pattern Recognition*, *76*, 569–581.

NordPool. (n.d.). *Day-ahead market.* https://www.nordpoolgroup.com/en/the-power-market/Day-ahead-market/.

Nowotarski, J., & Weron, R. (2018). Recent advances in electricity price forecasting: A review of probabilistic forecasting. *Renewable and Sustainable Energy Reviews*, *81*, 1548-1568. Retrieved from https://www.sciencedirect.com/science/article/pii/S1364032117308808 doi: https://doi.org/10.1016/j.rser.2017.05.234

NVE. (2021). *Wholesale market: Timeframes.* https://www.nve.no/norwegian-energy-regulatory-authority/wholesale-market/wholesale-market-timeframes/?ref=mainmenu.

Pascanu, R., Mikolov, T., & Bengio, Y. (2013). On the difficulty of training recurrent neural networks. In *International conference on machine learning* (pp. 1310–1318).

Phillips, P. C., & Perron, P. (1988). Testing for a unit root in time series regression. *Biometrika*, *75*(2), 335–346.

Salinas, D., Flunkert, V., Gasthaus, J., & Januschowski, T. (2020). Deepar: Probabilistic forecasting with autoregressive recurrent networks. *International Journal of Forecasting*, *36*(3), 1181–1191.

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

Smith, T. G., et al. (2017–). *pmdarima: Arima estimators for Python.* Retrieved from http://www.alkaline-ml.com/pmdarima ([Online; accessed 11/05/2023])

Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to sequence learning with neural networks. *CoRR*, *abs/1409.3215*. Retrieved from http://arxiv.org/abs/1409.3215

Svane, J. T. (2022). *Wind power forecasting as input to day-ahead trading strategies for wind in complex terrain* (Unpublished master's thesis). UiT The Arctic University of Norway.

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

Theodoridis, S., & Koutroumbas, K. (2009). Chapter 2 - classifiers based on bayes decision theory. In S. Theodoridis & K. Koutroumbas (Eds.), *Pattern recognition (fourth edition)* (Fourth Edition ed., p. 13-89). Boston: Academic Press. Retrieved from https://www.sciencedirect.com/science/article/pii/B9781597492720500049 doi: https://doi.org/10.1016/B978-1-59749-272-0.50004-9

TromsKraft. (n.d.). *Fakken vindkraftverk.* https://www.tromskraft.no/produksjon/kraftverk/fakken-vindkraftverk.

Tsai, Y. H., Bai, S., Yamada, M., Morency, L., & Salakhutdinov, R. (2019). Transformer dissection: An unified understanding for transformer's attention via the lens of kernel. *CoRR*, *abs/1908.11775*. Retrieved from http://arxiv.org/abs/1908.11775

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

UN, U. N. E. P. (2015). *Paris agreement*. Retrieved from https://wedocs.unep.org/20.500.11822/20830

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, *30*.

Wallace, J., & Hobbs, P. (2006). *Atmospheric science: An introductory survey*. Elsevier Academic Press. Retrieved from https://books.google.no/books?id=k4shngEACAAJ

Wang, Y., Hu, Q., Li, L., Foley, A. M., & Srinivasan, D. (2019). Approaches to wind power curve modeling: A review and discussion. *Renewable and Sustainable Energy Reviews*, *116*, 109422. Retrieved from https://www.sciencedirect.com/science/article/pii/S1364032119306306 doi: https://doi.org/10.1016/j.rser.2019.109422

Wang, Y., Zou, R., Liu, F., Zhang, L., & Liu, Q. (2021). A review of wind speed and wind power forecasting with deep neural networks. *Applied Energy*, *304*, 117766. Retrieved from https://www.sciencedirect.com/science/article/pii/S0306261921011053 doi: https://doi.org/10.1016/j.apenergy.2021.117766

Wangensteen, I. (2012). *Power system economics - the nordic electricity market (2nd edition)*. Fagbokforlaget.

Wright, L., & Demeure, N. (2021). Ranger21: a synergistic deep learning optimizer. *arXiv preprint arXiv:2106.13731*.