Tsunghao Huang

Comparison of Machine Learning Methods for Short-term Wind Speed Forecasting

Master Thesis

at the Chair for Information Systems and Statistics
(University of Münster)

Principal Supervisor:	Prof. Dr. Heike Trautmann
Associate Supervisor:	Dennis Assenmacher, M.Sc.
Presented by:	Tsunghao Huang [436499] Fischerweg 58 38518 Gifhorn +49 177 8564474 t_huan03@uni-muenster.de
Submission:	2nd July 2019

Contents

Figures III
Tables IV
1 Introduction 1
  1.1 Objective 2
  1.2 Thesis Outline 2
2 Problem Description 3
  2.1 Requirements 3
  2.2 Exploratory Data Analysis 3
    2.2.1 Time Series Decomposition 8
3 An Overview of Wind Speed Forecasting 12
  3.1 Multi-step-ahead Time Series Forecasting Strategies 13
4 Machine Learning 15
  4.1 Machine Learning Terminology 15
  4.2 Artificial Neural Networks 17
    4.2.1 Artificial Neuron 17
    4.2.2 Neural Network Structures 18
    4.2.3 Training Neural Networks 20
  4.3 Selected Models 25
    4.3.1 Baseline Models 25
    4.3.2 Neural Network Based Models 27
5 Experiments and Results 36
  5.1 Data Preprocessing 36
    5.1.1 Train Test Split 36
    5.1.2 Standardization 36
  5.2 ARIMA with exogenous variables 37
  5.3 Long Short-term Memory 40
    5.3.1 Grid Search for LSTM Structure 41
    5.3.2 Random Search for Learning Rate 42
    5.3.3 Residual Diagnostics 44
  5.4 Adaptive Neural Fuzzy Inference System 45
  5.5 WaveNet 46
    5.5.1 Grid Search for WaveNet Structure 46
    5.5.2 Random Search for WaveNet Learning Rate 47
    5.5.3 Residual Diagnostics 48
  5.6 Evaluation Metrics 48
  5.7 Results and Discussion 49
6 Conclusion and Outlook 55
Appendix 56
References 60

Figures

1 Pearson correlations 4
2 Wind rose 6
3 Timeplot of wind speed 7
4 Wind speed series decomposition 9
5 A McCulloch-Pitts neuron (Jain et al. 1996) 17
6 Activation functions 18
7 Feed-forward Neural Networks 19
8 Unrolled RNN 20
9 An example of early stopping 25
10 LSTM cell (Olah 2015) 28
11 ANFIS 30
12 Convolution 1D 32
13 Dilated causal convolution (Van Den Oord et al. 2016) 33
14 Residual block 34
15 Overview of WaveNet (Van Den Oord et al. 2016) 35
16 Train test split 36
17 Box-Cox Transformation 39
18 Differenced wind speed 40
19 ACF and PACF 40
20 Time series cross validation 41
21 LSTM structures 42
22 LSTM learning rate random search 43
23 LSTM residual diagnostics 44
24 WaveNet structures 47
25 WaveNet learning rate random search 48
26 WaveNet residual diagnostics 48
27 Step by step results comparison 51
28 Timeplot of test wind speed 52
29 Results of baseline models 53
30 Results of machine learning models 54
31 ARIMAX residual diagnostics 56

Tables

1 Variables . . . 3
2 Strengths of time series components . . . 10
3 Classification of forecasting methods by prediction horizon . . . 12
4 LSTM learning rate random search . . . 44
5 ANFIS results . . . 45
6 Prediction errors . . . 50
7 1 step prediction erorrs by directions . . . 50
9 LSTM 1st structure grid search . . . 57
11 LSTM 1st structure grid search with/without shuffle . . . 58
13 LSTM 2nd structure grid search . . . 59
15 WaveNet structure grid search . . . 60
17 WaveNet structure grid search with/without vdir . . . 61

1 Introduction

The annual wind power installations in EU have increased over the past years from 6.6 GW in 2005 to 15.6 GW in 2017, according to WindEurope (2018), and the power generation capacity from wind has become the 2nd largest in the EU since 2016. The steady growth in wind energy introduces high demands for better wind farm deployment planning, operation management, and wind turbine control.

Generally, a wind turbine control system has two optimization goals, namely maximizing power generation and minimizing structural loads and they are often contradictory to each other. To achieve these goals, model predictive control (MPC) and nonlinear model predictive control (NMPC) methods have shown potential improvement over the classic control methods due to their ability to include future information, which indicates that MPC/NMPC control systems could benefit from a precise short-term prediction of the relevant wind velocities (Henriksen 2010; Gosk 2011). Having the approximate estimation of the wind speed with a sufficient prediction horizon gives the turbine control system the capability to calculate the optimal adjustment of the wind turbine actuators with respect to future wind condition in the rotor plane. Furthermore, given the wind speed prediction, the control system would be able to reduce the mechanical solicitation of the wind turbine and therefore increase its lifetime. In other words, the benefits of the wind speed forecasting with sufficient prediction horizon helps to decrease the cost of maintenance of wind turbines and optimize the power generation.

Compared to traditional time series prediction models (e.g. AR, ARIMA), machine learning methods, especially neural network based models, have shown various advantages in modeling complex nonlinear function. Firstly, neural networks are able to automatically extract features from the raw data with little preprocessing. The ability has been demonstrated in several breakthroughs in many fields especially computer vision tasks using convolutional layers in neural networks (He et al. 2016; Krizhevsky et al. 2012; LeCun et al. 1998; Simonyan and Zisserman 2014). As for sequential data, a crucial preprocessing task is to make the time series stationary. Most of the traditional time series models require stationary input while neural networks might be able to extract features and model the non-stationary characteristic into the model with fewer assumptions. Secondly, while some extensions exist, in general, traditional time series models are not flexible with multivariate input data. Neural networks are flexible to incorporate both multivariate data and multi-output settings. These advantages have made neural network based methods promising in solving the problem of wind speed forecasting addressed by this thesis.

1.1 Objective

Currently, there are two main solutions for a turbine control system to predict wind speed. The first method is using a lidar wind speed measurement apparatus to scan the area in front of a wind turbine to estimate the wind speed beforehand (Smith and Harris 2007). The lidar sensor, however, is very expensive and it is constrained by the air quality. The other method is to assume wind speed will stay the same for the next couple of seconds. This is essentially the persistence model, which has the potential to be improved by more advanced methods. The objective of the thesis is to investigate whether machine learning models can better capture the future variations of wind speed against the persistence model with sufficient prediction horizon.

Three models are selected for implementation, namely Long Short-term Memory, Adaptive Fuzzy Inference System, and WaveNet. The three models are developed and compared based on the measurements that are customized according to the problem requirements. The result is expected to supplement the development of the wind speed forecasting model that is used by the wind turbine control system. The performance measurements of the result can provide an indication for further investigation of the forecasting model.

1.2 Thesis Outline

The structure of the thesis is as follow. First, the problem is introduced and an exploratory data analysis is performed on the given wind speed data in chapter 2. Next, chapter 3 gives a brief overview of the current status of the existing wind speed forecasting methods in the literature. This includes an introduction for the multi-step-ahead time series prediction strategies. The fundamental concepts of machine learning with a focus on the artificial neural network are introduced in chapter 4. An introduction of the selected methods is followed. Then, the detailed implementation and experiments results of the selected methods are presented in chapter 5. Finally, chapter 6 provides the conclusion and possible future research directions.

2 Problem Description

This chapter defines the specific requirements of the wind speed forecasting task and the results of exploratory data analysis (EDA).

2.1 Requirements

The discussion with the experts of wind turbine control system at IAV GmbH reveals that in order to fully make use of the prediction results, the control system needs precise forecasts with a sufficient prediction horizon and a reasonable resolution. Specifically, the ideal case is to have the prediction of the next 5-second wind speed every second.

The requirements lead to a **multi-step-ahead time series forecasting** problem. The variation of wind speed within the next 5-second is of interest rather than a single point wind speed estimation of 5-second later. In other words, the models should be able to forecast multi-step of wind speed in the next 5-second, so that the control system has sufficient knowledge about the future wind speed in order to counteract with it.

2.2 Exploratory Data Analysis

Name	Description
v_hub	Wind speed - main anemometer.
v_ref	Wind speed - reference anemometer.
v_mid	Wind speed - blade mid anemometer.
v_tip	Wind speed - blade tip anemometer.
vdir_hub	Wind direction - main vane.
vdir_ref	Wind direction - reference vane.
vdir_mid	Wind direction - reference vane.
T_hub	Main air temperature.
T_ref	Reference air temperature.
H_hub	Main relative humidity.
H_ref	Reference relative humidity.
P_hub	Air pressure.
Rain	Rain

Table 1 Variables

The available data provided by IAV GmbH consists of wind speed, wind direction, temperature, humidity, air pressure, and rain respectively. Since there is more than one sensor in different locations for the first 4 variables and there is no rain during the 6 days, there are 12 potentially usable variables in total. All sensors collected the data with a resolution of 1 Hz across 6 days. Table 1 gives a short description for each variable and fig. 3a shows a time plot for the scaled target variable of wind speed: v_hub. Note that due to the non-disclosure agreement, the data are standardized for visualization. Specifically, the data are scaled to standard score with the following formula

$$z = \frac{x - \mu}{\sigma}, \tag{2.1}$$

where x is the original data, μ and σ are the mean and the standard deviation of the samples respectively.

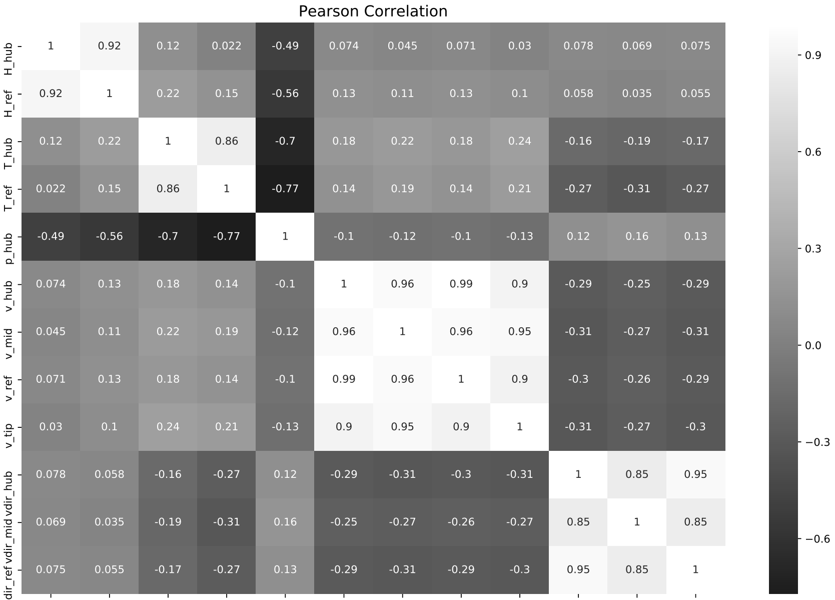


Figure 1 Pearson correlations

As expected, the correlation heatmap in fig. 1 shows that every main measurement (variable names with *_hub) is highly correlated with its references collected by different sensors. We can also see that the dependent variable, v_hub, has slightly negative correlations with the wind direction variables (vdir_*). This correlation might be influenced by the wind farm terrain. Further analysis is needed to investigate if there exist different characteristics of the wind from different directions. A wind rose graph might be able to give more meaningful insights by capturing the whole picture of wind speed and direction.

In general, except for the wind speed variables measured by different sensors, there are no strong correlations between v_hub and other variables. Furthermore, in addition to the wind speed signal itself and wind direction, other measurements such as pressure and temperature are rarely available for a turbine control system. Although including such information in the models might have the potential to improve the prediction results, training a model with additional measurements makes the assumption that these variables are always available. In order to better align with the real-world application, it is decided to focus on the bivariate scenario in the thesis.

The wind rose in fig. 2 gives an overview of the wind speed distribution along the wind direction. The length of each spike represents the probability of wind blows from the corresponding direction and each spike comprises different wind speeds which are indicated in different colors by the legend. One can see that the wind blows rarely from the southeast, consisting of only 7.56%. In addition, there is a spike from the southwest by south, indicating that the wind blows frequently from that direction.

In fig. 3b, based on four equal partitions, the wind speeds are categorized into cardinal directions, which are northeast (N-E), southeast (S-E), southwest (S-W), and northwest (N-W). The wind speeds from S-W have relatively higher variations compared to others while wind speeds from N-W have the smallest variance among the four categories. The differences in variance can be observed much easier if we detrended the wind speed by differencing the series, as shown in fig. 3c.

A wind turbine has several control actuators such as generator torque, blades pitch, and yaw angle adjustment. In the problem setting of this thesis, the yaw angle is controlled by the wind turbine controller. That is, we can assume that the turbine always faces to the primary direction of the wind. However, as mentioned above, there exist significant differences in the variance and distribution of wind speed coming from different directions. The differences are very likely to be influenced by external factors such as the wind farm terrain or climate properties. For example, there might be obstacles in the direction of southeast that block winds blowing from that direction but such information is unknown and can not be confirmed from the data. Nevertheless, by including the wind direction variable, we expect that the machine learning models might be able to derive the differences in wind speed coming from different directions.

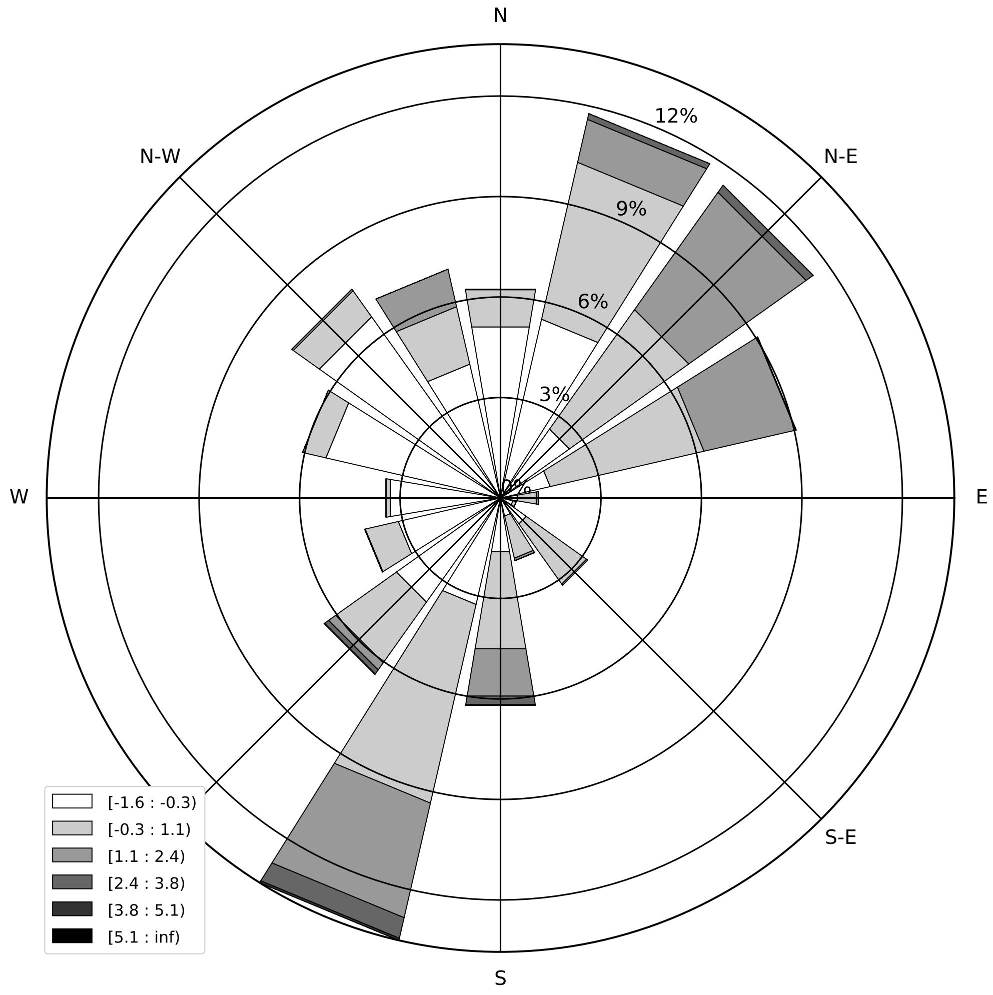
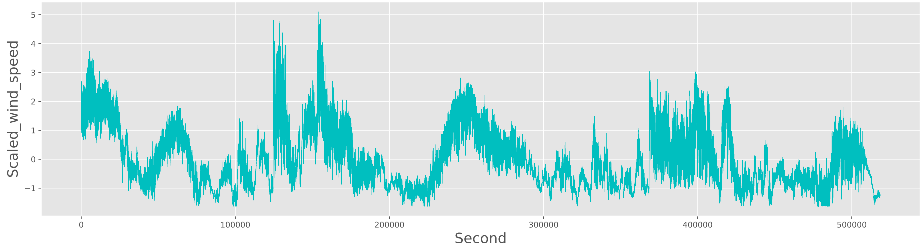
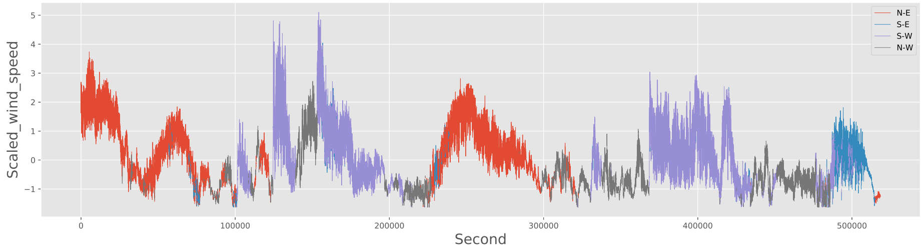


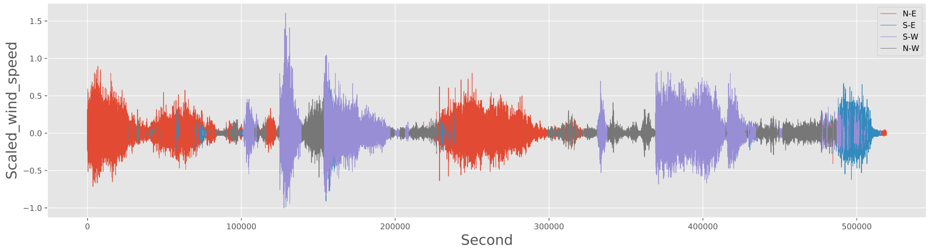
Figure 2 Wind rose



(a) Scaled wind speed



(b) Scaled wind speed colored by directions



(c) Detrended wind speed colored by directions

Figure 3 Timeplot of wind speed

2.2.1 Time Series Decomposition

Time series data consists of several patterns. Separating these patterns sometimes gives better intuition about the underlying function that generates the time series and thus implies the modeling choice. A time series data can be generally separated to trend, seasonality, cyclic, and residuals (Hyndman and Athanasopoulos 2018). A trend is a continuous increasing or decreasing pattern while seasonality represents the repetitive patterns exhibit in a fixed period. For instance, the air temperature usually has monthly seasonality, which means the variation of the temperature has a similar pattern across the year. The temperature in January 2019 is usually similar to the ones in January of different years. On the other hands, cyclic is the pattern that can be observed without a fixed period. This is the main difference between seasonality and cyclic. Lastly, residuals are the remained elements after extracting each of the other aforementioned components. According to Hyndman and Athanasopoulos (2018), the classical time series decomposition models assume that a time series can be decomposed into trend-cycle, seasonality components, and residuals. In an additive model, the time series can be represented as

$$y_t = S_t + T_t + R_t \tag{2.2}$$

where y_t is the original data, S_t is the seasonal component, T_t represents the trend-cycle component, and R_t is the residual which is left after the other components are removed. In a multiplicative model,

$$y_t = S_t \times T_t \times R_t \tag{2.3}$$

the components are multiplied to construct the original signal.

The decomposition starts with estimating the trend-cycle component from the original data. Classical time series decomposition models apply moving average on the observed time series to extract its trend-cycle component. Hyndman and Athanasopoulos (2018) suggested that the derived component, $\hat{T}_t$, can be written as

$$\hat{T}_t = \frac{1}{m}\sum_{j=-k}^{k} y_{t+j}, \tag{2.4}$$

where m is the period and $m = 2k + 1$, which means the estimated trend component $\hat{T}_t$ is the average of the period m that consists of y_t, the previous k steps and the next k steps. If using additive mode, based on equation (2.2), the detrended series can be calculated as $y_t - \hat{T}_t$ and the remains are seasonal and residual components, $S_t + R_t$. The seasonal component can be estimated by calculating the average patterns within the period m. For example, if given a monthly temperature data over several years, the seasonal component would be calculated by averaging the temperature of the same month across different years. In such case, the seasonal pattern is estimated using $m = 12$ since there are 12 months in a year. Finally, the residual can then be calculated by $R_t = y_t - \hat{S}_t + \hat{T}_t$.

To get a meaningful decomposition, the choice of period m plays an important role. Since the wind speed data is sampled in Hz, some common choice would be minutely, hourly, or daily. Fig. 4 shows a decomposition result of the wind speed variable, v_hub, by using a hourly period, which means $m = 60 \times 60 = 3600$. The detrended time plot shows that the variance of the wind speed changes over time, which can be also observed from fig. 3c. When using a traditional time series model, this implies that a transformation might be necessary to stabilize the variance. The other solution would be fitting the model with shorter input, then the variance would not vary that much in the shorter series.

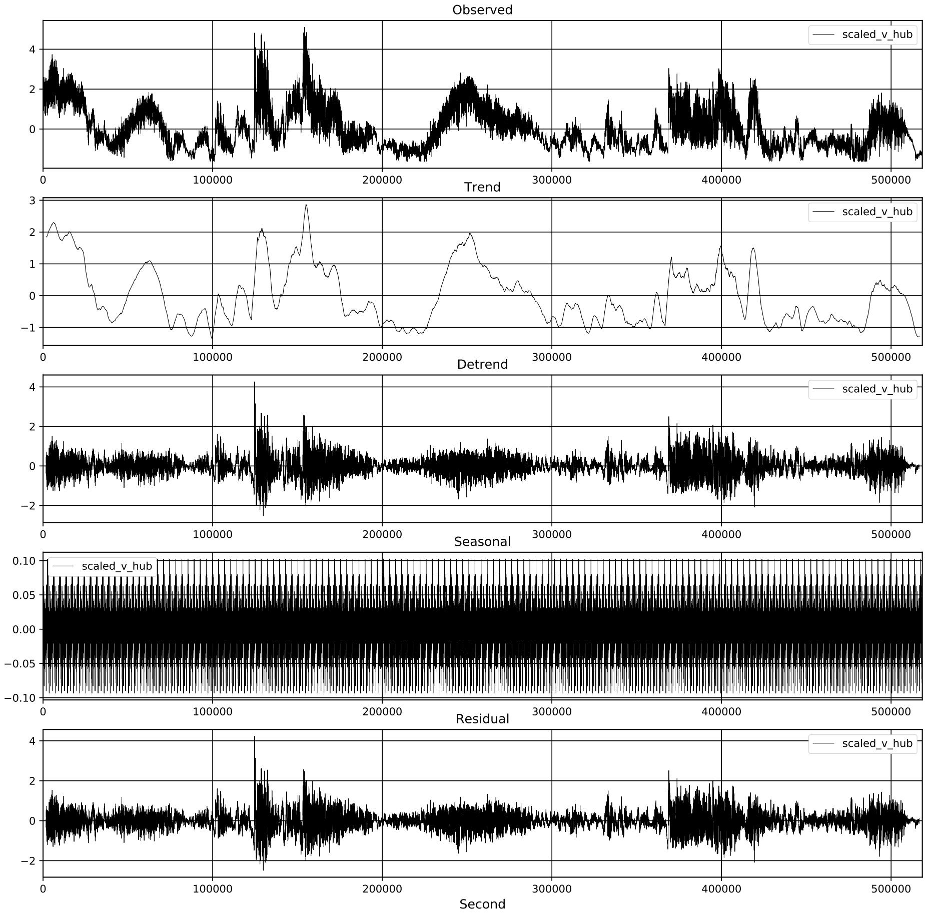


Figure 4 Wind speed series decomposition

The seasonal time plot comprises repetitive patterns with a period of one hour. As one can see, the residual does not show significant differences with the detrended series. This implies that the original signal does not contain significant hourly patterns. However, visual inspections might not be the best choice to evaluate the strength of seasonality and trend components exhibited in the original signal. Instead, Wang et al. (2006) suggested that the strength of trend F_T and seasonality F_S can be measured as

$$F_T = 1 - \frac{\mathrm{Var}(R_t)}{\mathrm{Var}(T_t + R_t)}, \tag{2.5}$$

$$F_S = 1 - \frac{\mathrm{Var}(R_t)}{\mathrm{Var}(S_t + R_t)}. \tag{2.6}$$

Recall that in equation (2.2), S_t is the seasonal component, T_t represents the trend, and R_t is the residual. Thus, $T_t + R_t$ represents the seasonally adjusted data, which can be calculated by $y_t - S_t = T_t + R_t$. If the original series contains strong trend-cycle components, then the variance of the seasonally adjusted data, $\mathrm{Var}(T_t + R_t)$, should be much larger than the variance of the residual, $\mathrm{Var}(R_t)$. According to equation (2.5), F_T would be closer to 1 in that case. The same intuition applies to F_S, if the original signal consists of strong seasonal components, then the variance of the detrended series, $\mathrm{Var}(S_t + R_t)$, would be much higher than the variance of the residual, $\mathrm{Var}(R_t)$, which results in a number closer to 1.

As shown in tab. 2, by calculating the trend and seasonal strengths, we can have a comparison of the strengths with different period m. In the case of the wind data that has a 1-second resolution, some common choices of m are 60, 3600, and 86400, corresponding to one minute, one hour, and one day.

m	F_S	F_T
60	0.000188	0.967497
3600	0.005852	0.859369
86400	0.225843	0.424240

Table 2 Strengths of time series components

The strength of the trend, F_T, would generally decrease with an increase of m. The intuition can be derived from equation (2.4). The larger the m parameter, the lower resolution of the estimated trend, which results in the decrease in F_T. Based on the results of F_T, the wind speed variable shows strong trended patterns. On the other hand, the small F_S

indicates that there are no apparent seasonal components related to the common minutely or hourly period in the original wind speed series data. There might exist slightly daily seasonal patterns but it is not that significant as the strengths of the trend. A periodogram can be helpful to identify the dominant periods when the time series data does not have significant seasonal patterns in the common period. However, there is also no clearly dominant period based on periodogram. Thus, we can conclude that the wind speed data is strongly trended and does not consist of strong seasonal patterns.

3 An Overview of Wind Speed Forecasting

Based on the prediction horizon, wind speed forecasting methods can be broadly classified into 3 groups, which are short-term (few seconds to minutes), medium-term (few hours), and long-term (days) (Jung and Broadwater 2014; Soman et al. 2010). The variation in the prediction horizon indicates different use cases. For example, the short-term wind speed forecasting is mainly applied for turbine control and dispatch planning. Tab. 3 lists some of the use cases for each category.

	Prediction horizon	Use case
short-term	seconds to minutes	Turbine control, Load tracking, Dispatch planning
medium-term	hours	Power system management
long-term	days	Maintenance scheduling

Table 3 Classification of forecasting methods by prediction horizon

Wind speed forecasting models can also be classified into physical and statistical models according to the modeling methods. Physical models are driven by processes. Building a physical model to forecast wind speed should follow the underlying physical laws and describe them using mathematical equations. On the other hand, statistical models mainly rely on external data without explicitly model the physics to capture the trend of wind speed (Foley et al. 2012; Giebel et al. 2011). Statistical methods can be further classified into traditional time series prediction models and machine learning based models.

Physical models tend to be used for forecasting wind speed over longer horizons ranging from one hour to days ahead (Tascikaraoglu and Uzunoglu 2014). Wind speed has high variation and is chaotic in the very short-term range (seconds). Due to the stochastic behavior, modeling the short-term wind speed with physical models involves complex physical equations. Without modeling the underlying physical laws with equations, statistical models are better at forecasting wind speed under such condition by learning the relationship between the past and future wind speed. Therefore, statistical models are considered to be the focus of the problem addressed by this thesis.

Machine learning methods are preferable in this thesis for several reasons. First, they make fewer assumptions of the input series while traditional time series models typically require the input series to be stationary, which is rarely the case in practice. Several preprocessing methods need to be done before configuring the model parameters. Second, machine learning methods, especially neural network based models, are flexible with input and output shape. In other words, they are flexible for multivariate and multi-step forecasting. The flexibility provides an advantage to tackle the problem defined in the thesis.

Despite many wind speed forecasting models are proposed in the literature, it is difficult to compare the proposed methods in the literature without actual implementation because there is no existing benchmark dataset. Most of the research evaluate the proposed methods on the different dataset with different metrics. Some use real measured data and some use synthetic data. Furthermore, the developed models are site-dependent, which means the model might only perform well on the specific wind farm location. This has made a throughout comparison difficult.

3.1 Multi-step-ahead Time Series Forecasting Strategies

Since the problem is defined as a multi-step-ahead time series forecasting problem, this section provides an introduction of the strategies used in this thesis, namely recursive and Multi-Input Multi-Output strategies.

Recursive strategy

In recursive strategy, the model is set to predict one step at a time and recursively use the one-step prediction as input to predict the next step until the prediction horizon is fulfilled. Formally, it can be written as

$$\hat{y}_{N+h} = \begin{cases} \hat{f}(y_N, y_{N-1}, ..., y_{N-d+1}), & \text{if } h = 1 \\ \hat{f}(\hat{y}_{N+h-1}, ..., \hat{y}_{N+1}, y_N, ..., y_{N-d+1}), & \text{if } h \in \{2, ..., d\} \\ \hat{f}(\hat{y}_{N+h-1}, \hat{y}_{N+h-2}, ..., \hat{y}_{N+h-d}),, & \text{if } h \in \{d+1, ..., H\} \end{cases}, \tag{3.1}$$

where d is the input size and H is the prediction horizons (Wang et al. 2016). Notice that the predictions are used recursively as inputs for further predictions. The recursive strategy provides the advantage to preserve the dependency among predictions. The downside is that the errors can be accumulated as the prediction horizon increases.

Multi-Input Multi-Output strategy

Multi-Input Multi-Output (MIMO) strategy simply uses one model to output a vector including the whole prediction horizon instead of a single step prediction. Using MIMO might reduce the risk of accumulated errors. However, it also has the disadvantage to reduce the prediction flexibility since all the points in the vector are predicted with the same model (Wang et al. 2016). MIMO strategy can be expressed formally as

$$[\hat{y}_{N+1}, \hat{y}_{N+12}, ..., \hat{y}_{N+H}] = \hat{F}(y_N, y_{N-1}, ..., y_{N-d+1}), \tag{3.2}$$

where H is the prediction horizon and d is the input size.

Including recursive and MIMO strategies, Wang et al. (2016) categorized 5 strategies for multi-step-ahead time series forecasting that are used in wind speed prediction. For instances, direct strategy tries to build a model for every prediction step and direct recursive strategy combines recursive and direct methods to produce the predictions. The other three strategies are not used because they require to train several models. For example, if the direct strategy is applied in the thesis, it would be necessary to build 5 different models for 5 prediction steps and each model have to go through its own hyperparameter optimization process. This is not ideal for neural network models since training such a model could easily take days or even weeks due to the complexity. Thus, the recursive strategy and MIMO strategy are used in this thesis.

4 Machine Learning

This chapter introduces the fundamental concepts of machine learning with a focus on artificial neural network (ANN). Section 4.1 explains the machine learning terminology that are used throughout the thesis. Section 4.2 introduces ANN concepts including the components in an artificial neuron and the algorithms that are required to train a neural network. Lastly, the selected baseline models and ANN-based models are presented in section 4.3.

4.1 Machine Learning Terminology

Machine learning is the study of statistics and algorithms for the computer system with the intentions to achieve a given task automatically. Depending on the feedback, three types of machine learning can be defined, which are supervised learning, unsupervised learning and reinforcement learning (Géron 2017).

Types of Machine Learning
In supervised learning, the model receives pairs of features/outcomes and tries to learn the underlying relationship. The actual outcomes provide feedback for the model to optimize its parameters. On the other hand, in unsupervised learning, only features are given for the model since the outcomes are not available. The model tries to directly infer the data structures and characteristics from the features. For example, clustering is a method that is often used in unsupervised learning to infer meaningful partitions of the data. In reinforcement learning, an agent learns to produce ideal outputs from a series of interactions with the environment. After the agent takes an action, it receives feedbacks indicating the performance of the action. Based on the feedback, the agent adjusts its next action.

The problem addressed in the thesis is considered as a supervised learning task since the outcome is known. The task is to learn a function that is able to map the input features to the ideal outcomes. Specifically, given n pairs of input-output data,

$$(x_1, y_1), (x_2, y_2), ..., (x_n, y_n),$$

where x_i is the feature vectors and y_i is the outcome of the i_{th} sample, the goal is to learn a function f that is able to generate the outcome, y_i so that $y = f(x)$.

Based on the output types, supervised learning can be further classified into classification and regression tasks. The output values of classification are within a finite set. For example, spam email detection can be seen as a binary classification problem since the model is trying to classify whether an email is a spam or not. For regressions, the output values can be any number within a range. This is the case for wind speed forecasting.

Loss Function

In supervised learning, the feedback is required to train a model. The loss function provides information on the model's performance during training. The loss function is the measurement between the desired outputs and the actual outputs of the model. With loss function, a model has the performance's measurement throughout the training phase. The goal of the optimization is to minimize the loss function by updating the parameters of the model. For instance, in the case of neural networks, the parameters would be the weights and biases. L_1 and L_2 loss functions are frequently used in machine learning. The two loss functions can be written as

$$L_1(y_i, \hat{y}_i) = \sum_{i=1}^{n} |y_i - \hat{y}_i|, \tag{4.1}$$

$$L_2(y_i, \hat{y}_i) = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2, \tag{4.2}$$

where n is the number of training samples, y_i and $\hat{y}_i$ are the true outcomes and the model's output for the i_{th} sample. As one can see, L_1 is used to express the sum absolute differences between the true and predicted values, while L_2 loss express the errors by the sum of squared differences.

Testing and Validation

The only way to get a genuine performance measurement of a model is to evaluate the new data that it has never encountered before. Since it is not practical to evaluate the model during production, the common method to get the generalized performance is to separate the available data into training and test set. The training data are used for optimizing the model and the test data is used for final evaluation. In general, test data should be only used for evaluation. It is not ideal to get the model performance on the test set during the training phase and then use the performance result to fine tune the model because the test data became known to the model in that case. A common solution is to hold out another set in the training data, called validation set. In that case, the model is trained on the training data without validation set and then use the validation data to get an intermediate performance measurement to optimize the model.

However, training an algorithm on a fixed dataset often yields too optimistic result. It might be possible that the model is over-fitting on the training data, which means the model performs well on the training data but cannot generalize on unseen data. Cross-validation is a technique widely used to solve this potential problem in building machine learning models. K-fold cross-validation randomly partitions the training data into k subsamples, train the model on $k-1$ subsamples, and evaluate the performance on the last 1 subsample (Arlot et al. 2010). The final performance evaluation would be the average of the k validation sets. This method prevents the model from overfitting on one particular dataset and gives a more generalized performance measurement. In section 5.3, cross-validation is adapted to the case of time series data and applied in the implementation for hyperparameters tuning.

4.2 Artificial Neural Networks

This section briefly presents the mathematical model of an artificial neuron and then the different types of network structures that are formed by the neurons.

4.2.1 Artificial Neuron

Artificial neurons are originally inspired by biological neurons in animals' brains. ANNs are computing systems consisting of large amounts of simple processors with interconnections (Jain et al. 1996). The first mathematical model of an artificial neuron, which is a binary threshold neuron, is proposed by McCulloch and Pitts (1943). As shown in fig. 5, the neuron computes the weighted sum of its input signals $x_1, x_2, ..., x_n$ and the threshold function outputs 1 if the summation is above the threshold u, or outputs 0 otherwise. Formally, a McCulloch-Pitts neuron can be written as

$$y = \theta\Big(\sum_{i=1}^{n} w_i x_i - u\Big), \tag{4.3}$$

where θ is the unit step function.

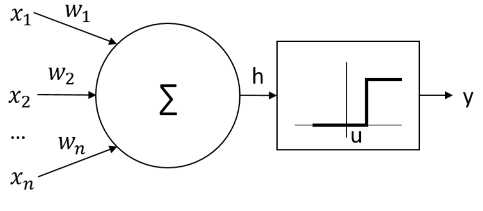


Figure 5 A McCulloch-Pitts neuron (Jain et al. 1996)

Activation Functions

In recent years, various activation functions are proposed to replace the threshold function used in the McCulloch-Pitts neuron. Sigmoid, Hyperbolic Tangent (tanh), and Rectified

Linear Units (ReLU) are three common activation functions and they are used in this thesis. The three functions can be written as

$$sigmoid(x) = \frac{1}{1+e^{-x}}, \tag{4.4}$$

$$tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}, \tag{4.5}$$

$$ReLU(x) = \begin{cases} x, & \text{if } x > 0 \\ 0, & \text{if } x \leq 0 \end{cases} \tag{4.6}$$

Fig. 6 shows a visualization of each activation function. One can see that the boundary for tanh is (-1, 1) and sigmoid is between (0, 1). ReLU only activates when the input is lager than 0. This is equivalent to $max(0, x)$.

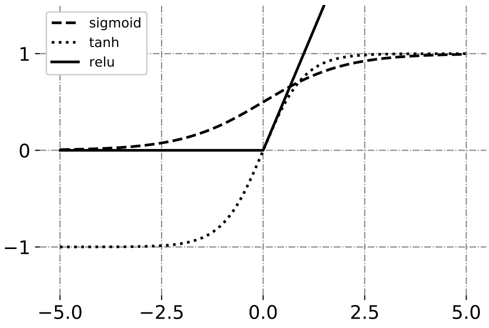


Figure 6 Activation functions

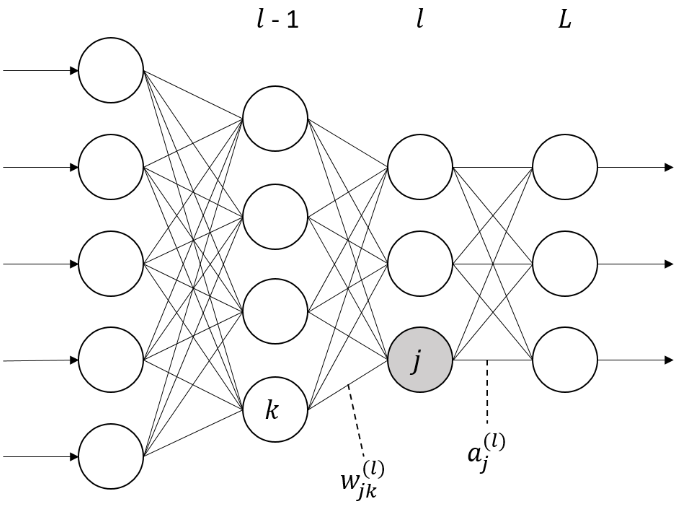
4.2.2 Neural Network Structures

In 4.2.1, the mathematical model of an artificial neuron is defined. Neural networks are formed by connecting these neurons with each other. Based on the directions of the connections, there are two types of neural networks, which are feed-forward and recurrent networks.

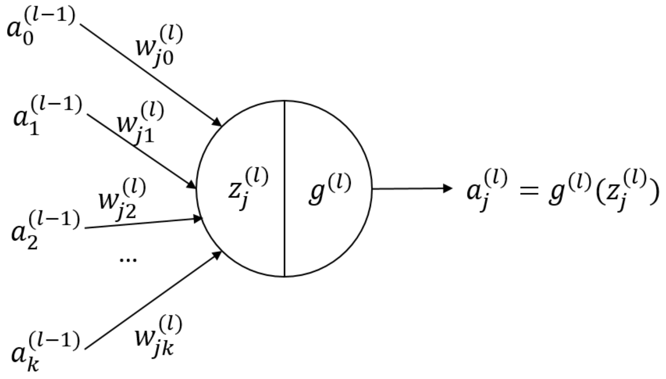
Feed-forward Neural Networks (FNN)

In ANN, the neurons are often organized in layers, each layer has several neurons. As the example in fig. 7a shows, there are four layers in this FNN, the first and the last layers are the input and output layers respectively. And the two layers in between are often referred to as hidden layers. The connections in Feed-forward Neural Networks (FNN) only send the signal in one direction, from the input layer to the output layer.

Each neuron has an activation function that introduces nonlinearity to the model. We can see the neural network as a composition of nested nonlinear functions and thus one can use ANN as a model for nonlinear regression (Russell and Norvig 2016, p.732).



(a) An example of a FNN



(b) A neuron in FNN

Figure 7 Feed-forward Neural Networks

Rcurrent Neural Networks (RNN)

Compared to a feed-forward neural network, the output of a neuron in a recurrent neural network (RNN) is fed back to itself for the next time step. In other words, a recurrent neuron gets two inputs, one from the current time step and the other from the previous. One can view this property as having a loop in the network. The loop enables the neurons to pass information from the previous to the future. Fig. 8 shows the unrolled view of a recurrent neural network. With this feature, RNN is able to look beyond the input data and to memorize information from the past. The ability is especially useful when dealing with sequential data, where the result of the current time step depends on the previous ones. The output h_t of the recurrent neurons can be written as

$$h_t = \tanh(x_t U + h_{t-1} W + b), \tag{4.7}$$

where b is the bais, U is the weights matrix for the input data x_t, and W is the weights for the output h_{t-1} from the previous time step. Tanh is the typical activation function used in RNN.

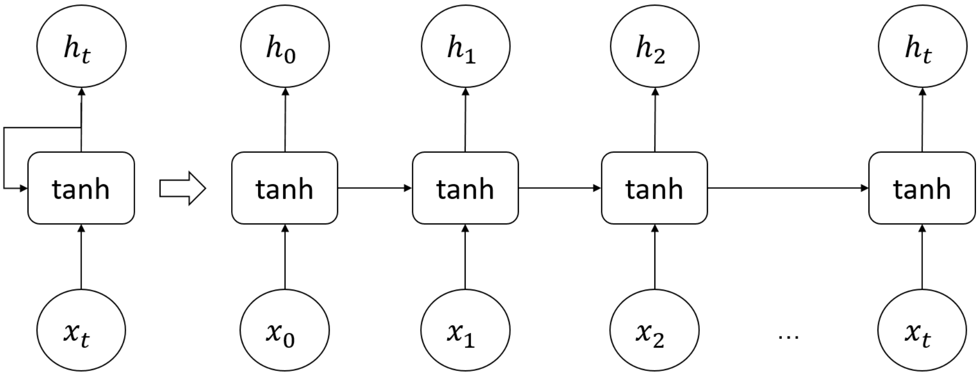


Figure 8 Unrolled RNN

4.2.3 Training Neural Networks

Training a neural network means adapting the network's parameters so that the neural networks can achieve better performance. To do that, it is necessary to define the model's performance with a loss function and optimize the function with algorithms such as gradient descent along with backpropagation. The explanation for these two algorithms is presented and the introduction of other methods that are used to train a neural network is followed.

Gradient Descent and Backpropagation

Weights and biases are the main parameters for a neural network. In order to optimize the model, the parameters need to be updated based on the training data. One of the most common optimization algorithms used to train neuron networks is gradient descent. The term, gradient, simply means the derivative of a function with several variables. For an optimization problem, it is required to have an objective function. This can be defined using the loss function introduced in section 4.1. In general, gradient descent works by iteratively update the weights and biases with one small step at a time in order to minimize the loss function (Nielsen 2015). The small changes are first calculated by the derivatives of the loss function w.r.t. each parameter. Then the actual size of the adjustment in every step is then multiplied by a small number called learning rate. Notice that the learning rate is often considered as a hyperparameter for the neural network. This means the value of the learning rate has to be decided beforehand since it is not trainable by the gradient descent algorithm. By continuously updating the parameters one step at a time, the weights and biases are going to be incrementally getting closer to the optimal values.

Backpropagation is developed and introduced by Rumelhart et al. (1988). It calculates the gradient of the loss function w.r.t. the weights with the chain rule. The calculated gradient is then used by gradient descent to update the correspondent weight in the neural network. The calculation is done in a backward fashion because the output of each layer depends on the weights and outputs from the previous layers. The general process is presented as pseudocode in the algorithm 1.

Algorithm 1 Backpropagation algorithm

```
Initialize model parameters (weights and biases)
while stop criterion not satisfied do
    Pass the input(s) forward to obtain the output(s)
    Compute the defined loss for all the output(s)
    for layer in layers do                          ▹ from the last to the input layer
        Calculate the gradients from current to the next layer
    Update weights
return network
```

The following paragraphs demonstrate calculation examples of backpropagation for one single weight in the output layer and one in the hidden layer. To get deeper into the calculation, it is necessary to first define some notations (Nielsen 2015). As illustrated in fig. 7, fig. 7b shows a neuron in fig. 7a. Based on the figures, we defined

- L is the number of layers in the network
- Layers are indexed by $l = 1, 2, ..., L - 1, L$
- Nodes in layer l are indexed as $j = 0, 1, ..., n - 1$
- Nodes in the previous layer $l - 1$ are indexed as $k = 0, 1, ..., n - 1$
- y_j is the desired output for node j in the output layer L
- $w_{jk}^{(l)}$ is the weight from node k in layer $l - 1$ to node j in layer l
- $w_j^{(l)}$ is the vector of weights connecting from layer $l - 1$ for node j in layer l
- $g^{(l)}$ is the activation function used in layer l
- $z_j^{(l)}$ is the weighted sum of node j in layer l,

$$z_j^{(l)} = \sum_{k=0}^{n-1} w_{jk}^{(l)} a_k^{(l-1)} \tag{4.8}$$

- $a_j^{(l)}$ is the activation output of node j in layer l,

$$a_j^{(l)} = g^{(l)}(z_j^{(l)}) \tag{4.9}$$

- C_0 is the loss function of the FNN for trainig sample 0,

$$C_0 = \sum_{j=0}^{n-1}(a_j^{(L)} - y_j)^2 \tag{4.10}$$

Notice that in equations (4.8) (4.9), and (4.10), C_0 is a function of $a_j^{(l)}$, $a_j^{(l)}$ is the activation output that depends on $z_j^{(l)}$ and $z_j^{(l)}$ is in turns a function of its input weights $w_j^{(l)}$. Thus the loss function can be expressed as a composition of functions

$$C_0 = \sum_{j=0}^{n-1} a_j^{(L)}\Big(g^{(l)}\big(w_j^{(l)}\big)\Big) \tag{4.11}$$

This is how one can use the chain rule to solve the partial derivative of the loss function w.r.t. one particular weight. For example, for a particular weight, $w_{jk}^{(L)}$ in the output layer, the partial derivative of the loss function to this weight can be written as

$$\frac{\partial C_0}{\partial w_{jk}^{(L)}} = \Big(\frac{\partial C_0}{\partial a_j^{(L)}}\Big)\Big(\frac{\partial a_j^{(L)}}{\partial z_j^{(L)}}\Big)\Big(\frac{\partial z_j^{(L)}}{\partial w_{jk}^{(L)}}\Big) \tag{4.12}$$

For the calculation of the weight, $w_{jk}^{(L-1)}$, in the hidden layer, one can apply equation (4.12) again to get

$$\frac{\partial C_0}{\partial w_{jk}^{(L-1)}} = \Big(\frac{\partial C_0}{\partial a_j^{(L-1)}}\Big)\Big(\frac{\partial a_j^{(L-1)}}{\partial z_j^{(L-1)}}\Big)\Big(\frac{\partial z_j^{(L-1)}}{\partial w_{jk}^{(L-1)}}\Big) \tag{4.13}$$

This is pretty similar to equation (4.12), however, notice that the first term in equation (4.13), the loss is not a direct function of the hidden activation output, which means the output $a_j^{(L-1)}$ in the hidden layer is not directly pass to the loss. Thus, it is necessary to apply the chain rule once again to replace the first term. This is the reason why the algorithm is called backpropagation. The algorithm repeatedly applies chain rule in a backward fashion to calculate the partial derivative of the weights in the hidden layer.

Types of Gradient Descent

Notice that until now the calculation only takes into account one training sample. Equation (4.12) is the partial derivative of the loss function w.r.t. the weight $w_{jk}^{(L)}$ for the first training sample. Based on how many samples are used to update the parameters, there are three types of gradient descent, namely stochastic gradient descent, batch gradient descent, and mini-batch gradient descent (Russell and Norvig 2016).

Batch size is a hyperparameter in gradient descent to control how many samples are used to update the parameters in order to minimize the loss function, while the number of epochs represents the number of times the complete training samples pass through the model. The number of epochs is another hyperparameter for gradient descent.

In stochastic gradient descent, the parameter is updated for every sample that passes through the model. This can be seen as setting the batch size equals to one for gradient descent. For instance, the update of a particular weight for the first sample can be written as

$$w_{jk}^{(l)} = w_{jk}^{(l)} - \alpha \frac{\partial C_0}{\partial w_{jk}^{(l)}}, \tag{4.14}$$

where α is the learning rate and C_0 represents the loss function for the first sample passes through the network.

For batch gradient descent, the parameters of the model being trained are updated with the whole training samples while the mini-batch gradient descent updates the model parameters with small batch size. Thus, the updated value for a particular weight can be written as

$$w_{jk}^{(l)} = w_{jk}^{(l)} - \alpha \Big(\frac{1}{n} \sum_{i=0}^{n-1} \frac{\partial C_i}{\partial w_{jk}^{(l)}} \Big), \tag{4.15}$$

where n is the batch size. If n equals to the training sample size, then equation (4.15) is a batch gradient descent. If n is between 1 and the training sample size, then the equation represents mini-batch gradient descent.

Vanishing and Exploding Gradient

Vanishing gradient often occurs when training artificial neural networks with gradient descent and backpropagation, especially the ones with a large number of layers. The weights that reside close to the beginning of the network has a high chance to be stuck without updating. The reason is that the gradient of the loss function w.r.t. particular weight is calculated using backpropagation and the algorithm calculates the gradient w.r.t. a particular weight with the chain rule.

As discussed earlier, by repeatedly applying chain rule, the gradient would become the product of derivatives that depend on the components that reside later in the network. Thus, the earlier in the network a weight is, the more terms would be needed in the product to get the gradient of the loss w.r.t. this weight. Notice that these terms consist of the derivatives of the activation functions. As introduced in subsection 4.2.1, the activation functions try to squish the input into a small number, depending on which activation function is used. This property makes the derivative of activation functions easily smaller than 1. For example, the derivative of the tanh activation function is between (0, 1). When many of the terms in the product are smaller than 1, the corresponding gradient is likely to be extremely small, which causes the gradient to vanish. Since the updated value of a particular weight is based on the product of the calculated gradient and the learning rate, the weight is barely going to move if the gradient is extremely small. The opposite of vanishing gradient is exploding gradient. This occurs when using the activation functions that have the possibility to take large derivatives. These two problems make training a neural network even more time consuming and difficult.

Early stopping

As discussed earlier in 4.1, a common approach to obtain a more generalized performance is to separate the data into three sets, which are training, validation, and testing set. Testing data is the set that should be used at the end of the optimization, while the training and validation sets are used during the training process. Early stopping is a regularization method that prevents a model from overfitting on the training data. Overfitting means that the model fits the training set too well but would not be able to generalize the performance to the unseen data.

During the training phase, a subsample of training data is held out as the validation set. The validation set is not used for training but for monitoring the loss of the model on the unseen data. Early stopping works by actively monitoring the training and validation loss and stop the training once the validation loss stops to decrease up to certain points (Bishop 2006, p.259). Due to its effectiveness and simplicity, early stopping is a widely used method for regularization in deep learning (Goodfellow et al. 2016, p.247).

In practice, the number of epochs and patience are two parameters that can be set to control the early stopping. The number of epochs controls the maximum times the whole training samples passing through the model during training, while the number of patience is set to monitor the training and validation loss. If the validation loss stops to decrease up to the predefined number of patience, the training would be stopped. Based on the setting, one can choose to recover the trained parameters from the epoch that has the lowest validation. Fig. 9 shows an example of a model's training process. The lowest validation loss happens at the 4th epoch and then start to increase afterward while the training loss continues to decrease until the end of the training process. This is a sign of overfitting. If early stopping is used during the training process and the patience is set to be 6, then the training would have stopped at the 10th epochs, which saves a lot of computation resources and prevents the model from overfitting.

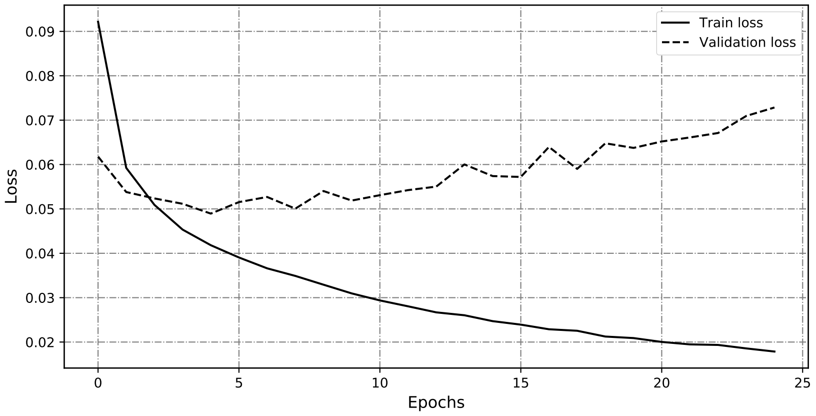


Figure 9 An example of early stopping

4.3 Selected Models

4.3.1 Baseline Models

Baseline models are typically easy and fast to implement. As its name suggests, the results of a baseline model are used as a threshold point to check whether a more complicated model is worth preserving. Two baseline models are selected for this thesis: persistence and autoregressive integrated moving average with exogenous variables (ARIMAX).

Persistence

For time series forecasting problem, the easiest possible model is the persistence model, where the predictions are simply equal to the last observation without modification. In section 2.1, the prediction horizon is set to be 5, so the predictions of a persistence model can be written as

$$\hat{y}_{t+h} = y_t, h = \{1, 2, 3, 4, 5\}, \tag{4.16}$$

where y_t is the observation of wind speed at time t and $\hat{y}_{t+h}$ is the prediction at time $t + h$.

ARIMA with exogenous variables

Although the main focus of this thesis is to evaluate the performance of machine learning models, it is worthwhile to compare the modeling methods with a traditional time series model. In addition, the process of modeling the time series with traditional methods provides insightful analysis that might be also useful when applying machine learning methods. As a result, the extension of ARIMA, ARIMA with exogenous variables, is selected as the other baseline model.

ARIMA model can be seen as a combination of the autoregressive (AR) model and the moving average (MA) model with differencing, which is represented as "integrated" in ARIMA. AR model assumes that the value of measurement at the next time step is a linear combination of the measurements from previous time steps, called lag values (Hyndman and Athanasopoulos 2018). The number of previous time steps used to predict the next measurement is indicated by the parameter p in the AR model and p is usually referred to as the order. Thus, an $AR(p)$ model can be written as,

$$y_t = c + \beta_1 y_{t-1} + \beta_2 y_{t-2} + \cdots + \beta_p y_{t-p} + \varepsilon_t, \tag{4.17}$$

where ε_t is the unpredictable white noise.

On the other hand, MA model uses the past forecast errors as the regressors to predict the next measurement (Hyndman and Athanasopoulos 2018). As in the AR model, a parameter q controls how many past forecast errors should be included. A $MA(q)$ can thus be written as

$$y_t = c + \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + \cdots + \theta_q \varepsilon_{t-q} + \varepsilon_t, \tag{4.18}$$

where ε_t is the white noise.

As mentioned, the ARIMA model is a combination of AR and MA model with differencing. Differencing is a method used in the time series model to achieve stationarity. For a time series to be stationary, it cannot has time-dependent components. That means time series with trend or seasonality components cannot be considered as stationary. Formally, Hyndman and Athanasopoulos (2018) defines that if a time series y_t is stationary, then the distribution of $(y_t, y_{t+1}, ..., y_{t+s})$ does not depend on t.

Both the trend and seasonality components can be removed by differencing. In ARIMA, the differencing times is indicated by the parameter d. Mathematically, the 1st order differenced series can be written as

$$y'_t = y_t - y_{t-1}, \tag{4.19}$$

while the 1st order seasonality differencing can be written as

$$y'_t = y_t - y_{t-m}, \tag{4.20}$$

where m is the identified period for seasonality. This can be integrated as an additional parameter for ARIMA, which is called seasonal ARIMA (SARIMA). However, as discussed in 2.2.1, there is no dominant seasonality components in the wind speed series so SARIMA is not applied.

Based on the descriptions above, an ARIMA model has three hyperparameters (p, d, q), where p is the order of the AR model, d is the order of differencing, and q represents the order of the MA model. Thus, the ARIMA model can be expressed as

$$y'_t = c + \beta_1 y'_{t-1} + \cdots + \beta_p y'_{t-p} + \theta_1 \varepsilon_{t-1} + \cdots + \theta_q \varepsilon_{t-q} + \varepsilon_t, \tag{4.21}$$

where y'_t is the differenced series.

Although traditional time series models usually focus on the univariate scenario, one extension for ARIMA does exist. That is the ARIMA model with exogenous variables (ARIMAX). The extension enables ARIMA to include external variables as additional regressors for model fitting and forecasting. To do this, ARIMAX simply adds the exogenous variable in the ARIMA equation,

$$y'_t = c + \phi x_t + \beta_1 y'_{t-1} + \cdots + \beta_p y'_{t-p} + \theta_1 \varepsilon_{t-1} + \cdots + \theta_q \varepsilon_{t-q} + \varepsilon_t, \tag{4.22}$$

where x_t is the external variable at time t and ϕ is its coefficient Hyndman and Athanasopoulos (2018).

The process of identifying ARIMAX hyperparameters following the aforementioned steps is presented in section 5.2.

4.3.2 Neural Network Based Models

Three artificial neural network based models are selected for the experiments, which are long short-term memory, adaptive neural fuzzy inference system, and convolutional neural network, respectively. A brief introduction and the reasons for selection are given in this section.

Long short-term memory

In theory, RNN is able to store the inputs from the previous and learn long-term dependencies between different time steps to produce the ideal outcome. However, in practice, RNN suffers from the problem of vanishing gradient (Hochreiter 1998).

As introduced in 4.2.3, the more layers a neural network has, the more likely it suffers the problem of vanishing gradient. This is because backpropagation has to apply more chain rules to calculate the gradient of the parameter that resides close to the input layer. The vanishing gradient also appears in RNN, which uses backpropagation through time algorithm to update its weights (Bengio et al. 1994). In general, as fig. 8 shows, each timestep of the input can be seen as an additional layer of the RNN when unrolled. This indicates that during training, the backpropagation algorithm has to propagate the error through these unrolled layers. Thus, as the timesteps increase, the gradient of the weight at the beginning would also suffer from vanishing because the terms in the calculation of gradient increase with the timesteps. It is the vanishing gradient problem that makes it difficult to train the RNNs.

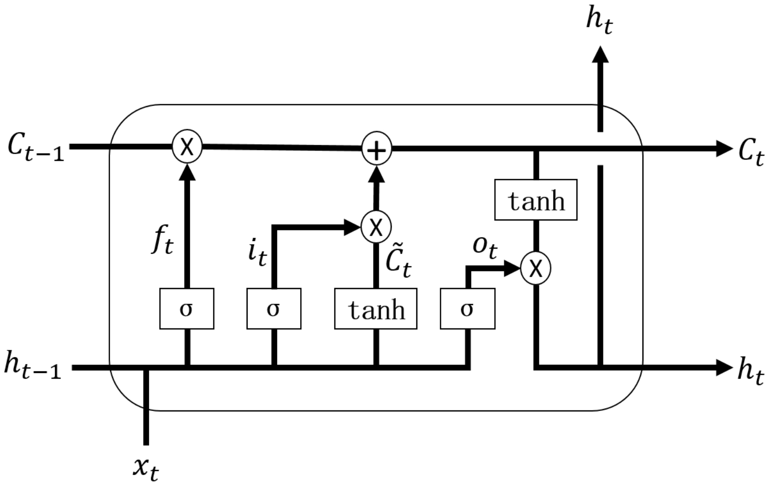


Figure 10 LSTM cell (Olah 2015)

Fortunately, a special type of RNN, Long short-term memory (LSTM), came to rescue. Instead of using a simple hyperbolic tangent in the recurrent neuron, LSTM introduces the state information flow C_t and 3 gates to control the outputs to of the neuron (Hochreiter and Schmidhuber 1997).

$$i_t = \sigma(x_t U_i + h_{t-1} W_i + b_i) \tag{4.23}$$

$$f_t = \sigma(x_t U_f + h_{t-1} W_f + b_f) \tag{4.24}$$

$$o_t = \sigma(x_t U_o + h_{t-1} W_o + b_o) \tag{4.25}$$

Similar to a basic recurrent neuron, the neuron in LSTM gets the input from the current step, x_t, and the output from the last step, h_{t-1}. Additionally, the neuron gets a third input, C_{t-1}, which is the state from the last time step. The first two inputs, x_t and h_{t-1}, go through the sigmoid activation function to form the "forget gate", f_t. Since the output of a sigmoid function is between 0 and 1, this makes the activation function a natural choice for gating. This can be seen as a mechanism to control what information the neuron would keep or forget from the last time step. On the other hand, the input gate controls the information that is added to the new sate, C_t. As equations (4.23) and (4.26) show, x_t and h_{t-1} flow through a sigmoid and a hyperbolic tangent function respectively and multiply together to get the additional information that is going to be added to the new state in equation (4.27), where the state is updated. Finally, in equation (4.28), the new cell state, C_t, goes through another hyperbolic tangent function and multiply by the output gate, o_t, to produce the other output of the neuron, h_t. The whole information flow is visualized in fig. 10.

$$\tilde{C}_t = \tanh(x_t U_g + h_{t-1} W_g + b_g) \tag{4.26}$$

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{4.27}$$

$$h_t = \tanh(C_t) * o_t \tag{4.28}$$

The introduction of the state information flow, C_t, and the gated functions make it possible for LSTM cell to control the gradient. Due to its ability to overcome the vanishing gradient problem, LSTM netowrk has demonstrated promising results in several applications, where the input data are sequential in natural. For instances, speech recognition (Graves et al. 2013), music generation (Oore et al. 2017), sentiment classification (Tang et al. 2015), meachine translation (Luong et al. 2014), and name entity recognition (Chiu and Nichols 2016), to name just a few.

Since wind speed modeling is also a sequential learning task, LSTM naturally becomes one of the popular choices to apply for many research in wind speed prediction. Kulkarni et al. (2019) compares LSTM and nonlinear autoregressive network with external inputs (NARX) for long-term wind speed prediction. They conclude that LSTM performs better and is more efficient than NARX for their use case, which is turbine blade design. Zaytar and El Amrani (2016) trained several multi-stacked LSTM models with 15 years of weather data for 9 cities in Morocco. The results suggest that LSTM is comparable to traditional time series model and it can be considered as a better solution for general weather prediction problem. Additionally, LSTM is combined with other models to forecast wind speed. In Qin et al. (2019), Hilbert–Huang transform and fuzzy entropy are used before LSTM to predict the wind speed. The input wind speed data is first decomposed into several components, classified according to the fuzzy entropy, and then fed into a group of LSTMs for prediction. Their experiments show that the proposed combined model performs better than using LSTM alone. A combination of convolutional neural network and LSTM is proposed in (Wu et al. 2016) The convolution layer is applied first to provide feature abstractions and the outputs of convolution are then fed into LSTM layers.

Adaptive neural fuzzy inference system

In adaptive neural fuzzy inference system (ANFIS), the components of the fuzzy system are integrated with neural network structures to leverage the learning ability of neural networks. Fuzzy logics provide algorithms with the option to transform the linguistic rules that are based on domain knowledge into automatically learned strategies. The introduction of fuzzy rules enables a certain degree of uncertainty since the "crisp" rules are converted to membership functions that allow partial membership. By combining neural networks, the system can automatically learn the membership functions by tuning the parameters and thus adopt different functions accordingly.

ANFIS has been used in wind speed/power estimation. Shamshirband et al. (2014) proposed ANFIS for wind speed estimation in wind power generation systems, however, they adopted a different approach by using wind turbine power coefficient, rotational speed and blade pitch angle as the input variables rather than the wind speed itself. Castellanos and James (2009) compares univariate and multivariate models to predict 1-step (hour) ahead wind speed with input variables wind speed and air pressure. Potter and Negnevitsky (2006) used ANFIS to predict 1-step (2.5 minutes) ahead wind speed and achieve a better result than the persistent model.

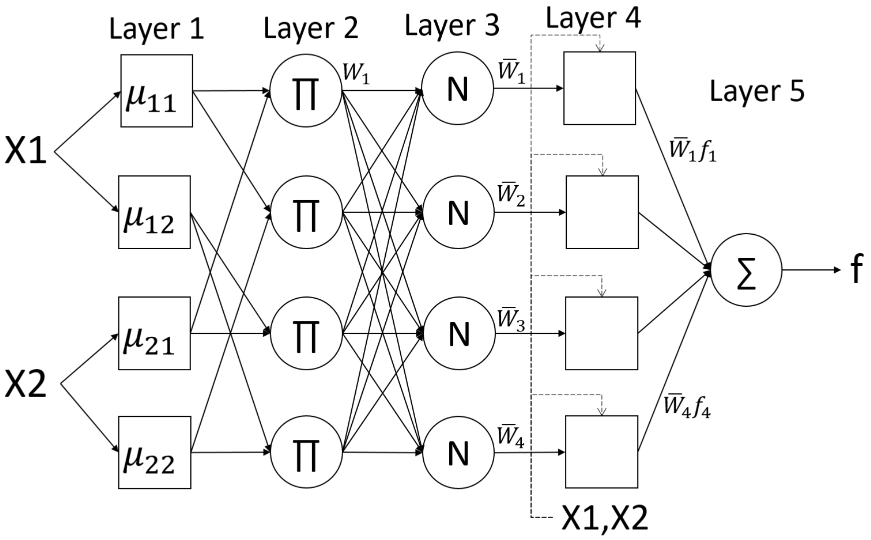


Figure 11 ANFIS

ANFIS consists of 5 layers. Instead of using activation functions in the typical neural network, the nonlinearity is introduced in the first layer, where the input data has to pass membership functions to compute the degree of memberships. The values are then served as the base for further computation for the normalized weights in the later layer. The detailed description of the ANFIS model is given below.

Layer 1: Every node in this layer can be seen as a membership function adapting to the function parameter. The output from each node is a degree of membership value corresponding to a fuzzy rule. According to Jang (1993), assuming the model has input features $x_i, i = 1, 2, 3, ..., n$ and every input has K rules, then the membership function can be written as a generalized bell function

$$\mu_{ij}(x_i) = \frac{1}{1 + [(\frac{x-c_{ij}}{a_{ij}})^2]^{b_{ij}}}, j = 1, 2, 3, ..., K \tag{4.29}$$

or a Gaussian function

$$\mu_{ij}(x_i) = exp[-(\frac{x - c_{ij}}{a_{ij}})^2], j = 1, 2, 3, ..., K \tag{4.30}$$

respectively, where μ_{ij} is the j_{th} rule for input x_i a_{ij}, b_{ij}, c_{ij} are the parameters for the membership functions. The shape of the membership function changes accordingly with the parameters. Since the parameters are part of the overall network structure, the parameters can be learned by usual neural network optimization algorithms such as backward propagation.

Layer 2: The output node is the result of the multiplication of the input. For instance, w_1 in fig. 11 can be written as

$$w_1 = \mu_{11}(x_1) \times \mu_{21}(x_2). \tag{4.31}$$

Note that the weight is the multiplication of degree of membership from different input, which means the model would end up with K^n rules

Layer 3: Weights normalization layer. The firing strengths resulting from the rule combinations are calculated in this layer.

$$\bar{w}_i = \frac{w_i}{\sum_{i=1}^{K^n} w_i}, i = 1, 2, 3, ..., K^n \tag{4.32}$$

Layer 4: This layer is referred to as the consequent in the original paper Jang (1993), which is the adoption of the Takagi and Sugeno's fuzzy if-then rule (Takagi and Sugeno 1993). Every node receive all inputs x_i. A node in this layer can be seen as a linear function multiplied by the weighted firing strengths from the last layer. For instance, in fig. 11, the output of the first neuron in layer 4 can be written as

$$\bar{w}_1 f_1 = \bar{w}_1(p_1 x_1 + q_1 x_2 + b_1), \tag{4.33}$$

where p_1, q_1, b_1 are the weights and bias for the linear function.

Layer 5: Summation layer. The output of this layer is simply a summation of all the input signals from layer 4. Thus, the overall output can be written as

$$\sum_{i=1}^{K^n} \bar{w}_i f_i = \frac{\sum_i w_i f_i}{\sum_i w_i}, \tag{4.34}$$

where f_i is the linear functiion that is specified in layer 4.

WaveNet

Before introducing WaveNet, it is important to know the basics about the convolutional neural network (CNN) first because WaveNet is basically a CNN with special features. Convolution is an important concept that is used in signal and image processing for computer vision, which means there exist 1D, 2D, or higher dimensions convolution. However, since the scope of the thesis is within time series forecasting, only 1D convolution is introduced here. Generally, one can think of the convolution operation as a filter (kernel) that slides across the input signal step by step to produce output. For each input position the filter sliding through, it calculates the dot product of the corresponding inputs with its weights. For example, in fig. 12, a filter of length 2 is sliding through the input signal one step at a time. The weights of the filter are $[0.5, 0.5]$. One can notice that the output size shrinks by 1 after convolution. This can be solved by adding a 0 at the beginning of the sequence so that the output sequence would have the same length as the input.

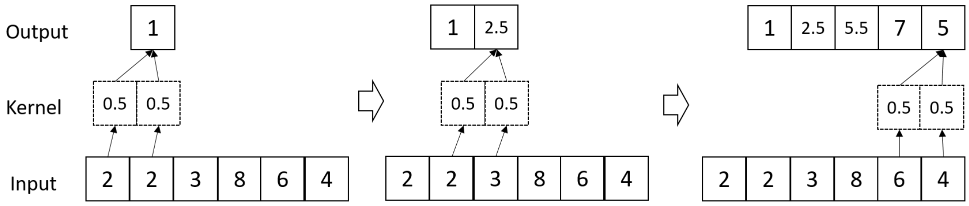


Figure 12 Convolution 1D

Convolution is a common method for computer vision. Before neural networks became popular, the weights of filters typically have to be designed by the scientists in order to get the desired outputs such as edge detection, blurring, and sharpen, etc. Due to the development of computational power, training deep neural network with optimization algorithms became more feasible. Scientists have incorporated convolutional operations in neural networks layer to solve various computer vision tasks. The weights of the convolutional layer can be automatically adjusted by optimization. Most of the recent breakthroughs apply convolutional layers in their structures (He et al. 2016; Krizhevsky et al. 2012; LeCun et al. 1998; Simonyan and Zisserman 2014).

In the field of wind speed forecasting, there is relatively fewer research applying CNN. Huang and Kuo (2018) builds a neural network that incorporates convolution and fully connected layer. The model is trained using 7 days of hourly data and evaluate on the next 3 days. The results show that the model is able to capture the hourly trend of wind speed. In terms of sequence learning in general, CNN is getting more attention recently despite no recurrent connections for its neurons. For example, Gehring et al. (2017) proposed a model structure that is entirely based on convolutional neural network and found that the model outperforms the LSTM model on benchmark machine translation datasets.

CNN has also shown significant improvements in speech generation. Van Den Oord et al. (2016) proposed WaveNet, a generative convolutional neural network, which is able to generate more natural human speech by directly modeling the raw waveform audio signal. The results show that the speech generated by WaveNet beat state-of-the-art algorithms based on a subjective evaluation.

Instead of using normal convolutional layers in the model, the authors apply the dilated causal convolution layer, which is the core component of WaveNet. In the autoregressive model, the prediction is based on the previous values. However, in normal 1-dimensional convolution, it is possible to include future values to predict the future. Causal convolution makes it possible to ensure that no future information is used for the prediction. That is, a causal convolution filter at time t can only operate on the inputs that are no later than t (Bai et al. 2018). Causal convolution provides solutions for modeling temporal sequence but one of the problems is that it requires additional layers to increase the receptive fields. To address this problem, WaveNet uses dilated convolution to increase larger receptive fields with fewer layers and parameters. The dilated rate is used to define the skipped steps in the filter. It can also be seen as using a larger convolutional filter but replacing the weights with 0 by the dilated rate (Bai et al. 2018). With the introduction of dilated convolution, it is possible for the model to increase its receptive fields exponentially with the numbers of layers. For instance, in fig. 13, with only 4 layers, the prediction can include 16 time steps into its inputs.

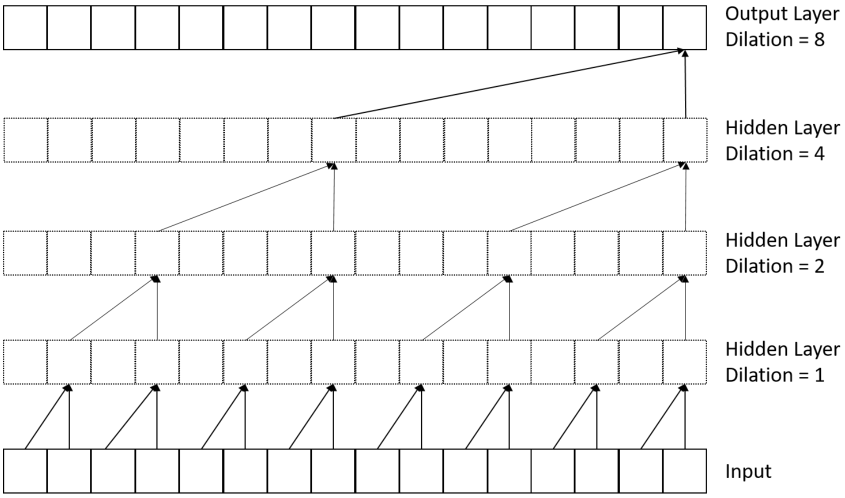


Figure 13 Dilated causal convolution (Van Den Oord et al. 2016)

After each causal dilated layer, Van Den Oord et al. (2016) propose to use the gated activation function instead of the rectified linear activation function, which is normally used in the convolutional layer. The suggestion is based on the results of their experiments, which indicates that the former works significantly better. Mathematically, the activation function can be written as

$$z = tanh(W_{f,k} * x) \odot \sigma(W_{g,k} * x), \tag{4.35}$$

where k is the index of layer, f and g represent filter and gate respectively, and W is the weight of the convolutional filter. As one can notice that this gated activation is similar to the gates in the LSTM cell.

Lastly, to train deeper neural network structures and overcome the possibility of vanishing or exploding gradient, both residual and skipped connection are used in WaveNet (Van Den Oord et al. 2016). Residual learning enables deep learning scientists to train even deeper neural networks by taking the activation output from one layer and feeding it to another layer that is even deeper in the structure (He et al. 2016). The main ingredient of a residual network is the residual block. Fig. 14 gives an example. Originally, without the residual connection (dotted line), the output of layer $l + 2$, $a^{[l+2]}$, can be calculated as

$$a^{[l+2]} = g(W^{[l+2]}a^{[l+1]} + b^{[l+2]}), \tag{4.36}$$

where g is the activation function, W is the weights matrix, b is the bias. With the residual connection, $a^{[l]}$ skip 1 layer and feed further to the input of layer $l+2$. Thus, the new $a^{[l+2]}$ can now be written as

$$a^{[l+2]} = g(W^{[l+2]}a^{[l+1]} + b^{[l+2]} + a^{[l]}). \tag{4.37}$$

Notice that the connection feeds to the input of layer l+2 but before the activation function. With the help of residual ingredient in the network, He et al. (2016) shows that, compared to the plain neural network, training error can be continuously reduced by increasing the layers.

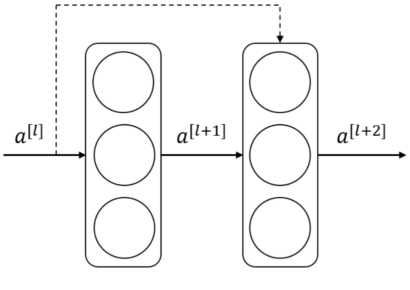


Figure 14 Residual block

Skip connection is a closely related concept as the residual network. In deep neural networks, one can think of each hidden layer as a different level of inputs abstraction or feature map. Because of the design of normal neural networks, the input data has to pass through a sequence of layers to finally get to the output layer. However, sometimes, it is difficult for the networks to figure out the desired output only based on the highest level of abstraction. To keep the benefits of both low and high levels of abstraction, skip connections introduce extra connections between layers that skip one or more layers (Orhan and Pitkow 2017). In addition, skip connections provide more direct paths for backpropagation algorithms to calculate the gradients, especially in deep neural networks. In WaveNet, the authors connected the outputs of each convolutional layer before the last two 1×1 convolutions. The overall structure of WaveNet is shown in 15. Notice that WaveNet uses Softmax as the final layer to produce a distribution of the output. However, it is decided to directly use the last 1×1 convolutional layer as outputs in this thesis.

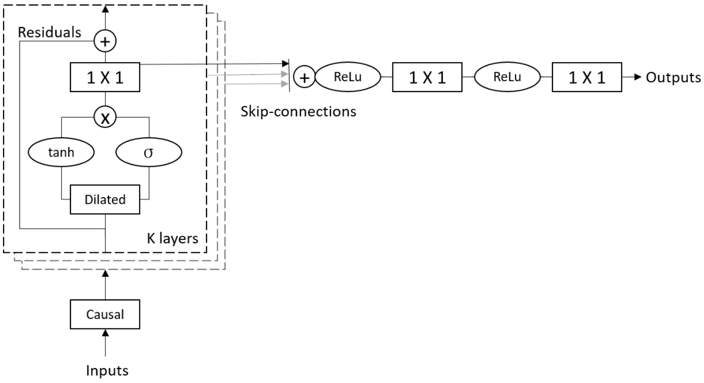


Figure 15 Overview of WaveNet (Van Den Oord et al. 2016)

There are several reasons to choose WaveNet as one of the machine learning models for comparisons. First, compared to normal convolutional layer, the dilated causal convolution makes it possible to include large amounts of lag values (receptive fields) with less computation power. Second, the model is designed to be autoregressive, which means the prediction of the model is based on the previous values. The model can directly model the raw waveform signal. This matches the settings of other selected models and fulfills the requirements of the problem, where the future wind speed is predicted based on the previous. Last but not least, the use of residuals and skip connections has the potentials to capture different levels of features in the wind speed series.

5 Experiments and Results

This chapter describes the implementation of the models regarding data preprocessing, structure optimization, and parameters selections. An introduction of the evaluation metrics used in the thesis is presented before showing the result and discussion section.

5.1 Data Preprocessing

This section introduces the data preprocessing methods used for the selected models in general. However, ARIMAX used has its own preprocessing process to follow. This is introduced in the correponding section 5.2.

5.1.1 Train Test Split

As discussed in 4.1, before implementing and training the models, it is important to separate the training and testing data in order to provide an unbiased evaluation of the performance. Due to the temporal characteristics of time series data, the separation has to preserve the order of the series. To fulfill the aforementioned requirement, it is decided to use the first 80% of the series as training data and the following 20% as testing data. Fig. 16 shows the split point of training and test set of the wind speed series. The selected models are prepared on the training set and evaluated on the testing set.

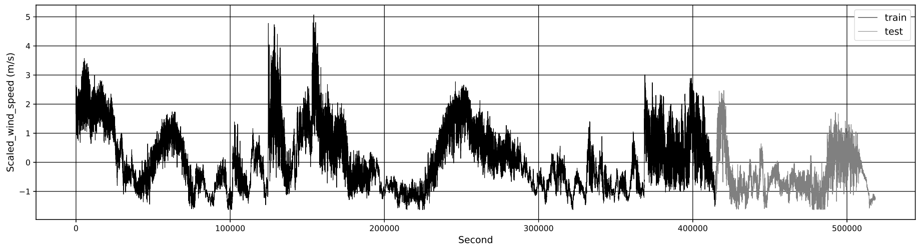


Figure 16 Train test split

5.1.2 Standardization

Since neural network models use activation function to introduce nonlinearity to the model, it is important to scale the data before training the model. To make sure that the models do not know the testing data beforehand, training and test data are scaled separately with the standard score. This is introduced in equation (2.1) Note that ARIMAX has its own preprocessing methods that need to be followed. Therefore, standardization is only applied for the neural network based models.

5.2 ARIMA with exogenous variables

As mentioned in chapter 2, it is decided to use wind speed and wind direction as inputs for the selected model. The following process of ARIMAX implementation includes fitting the exogenous variable, wind direction.

Selecting the proper orders for ARIMAX can be tricky. The order of d is mainly used to make the time series stationary while the orders q and p can generally be identified by checking the autocorrelation function (ACF) plot and partial autocorrelation function (PACF) plot. The autocorrelation is the correlation between the measurement with itself at different time steps. According to Hyndman and Athanasopoulos (2018), the autocorrelation function can be represented as

$$r_k = \frac{\sum\limits_{t=k+1}^{T} (y_t - \bar{y})(y_{t-k} - \bar{y})}{\sum\limits_{t=1}^{T} (y_t - \bar{y})^2}, \tag{5.1}$$

where r_k is the correlation between y_t and y_{t-k} and T is the length of the time series.

By definition, the autocorrelation of an $MA(q)$ process is zero at lag $q + 1$ and beyond (Shumway and Stoffer 2017, p.90). That is the main reason to use ACF plot to select the possible q. However, ACF plot is not the ideal tool to choose the order p because there might be redundant correlations in between. For instance, in addition to direct correlation, the autocorrelation r_3 might also include indirect effects from r_2 and r_1. Partial autocorrelations address the problem by removing the autocorrelation of other lag values in between. Thus, the PACF plot is used to select the proper order p of the AR model.

Box–Jenkins method (Box et al. 2015; Hyndman and Athanasopoulos 2018) provides general procedures to follow when applying ARIMA. The steps are listed below:

(1) Identify time series components.

(2) Stabalize the variance if necessary.

(3) If necessary, difference the time series to achieve stationarity.

(4) Identify possible model parameters by looking at the ACF and PACF plots.

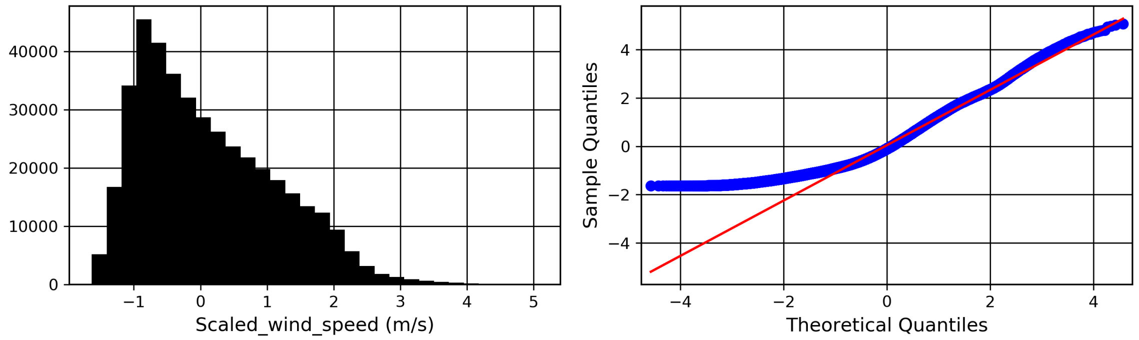
(5) Compare the candidate models by information criteria, e.g. Akaike information criterion or Bayesian information criterion, to get a better model.

(6) Check the residuals' timeplot and ACF plot to see if the residuals are white noise. If not, repeat the steps from (4).

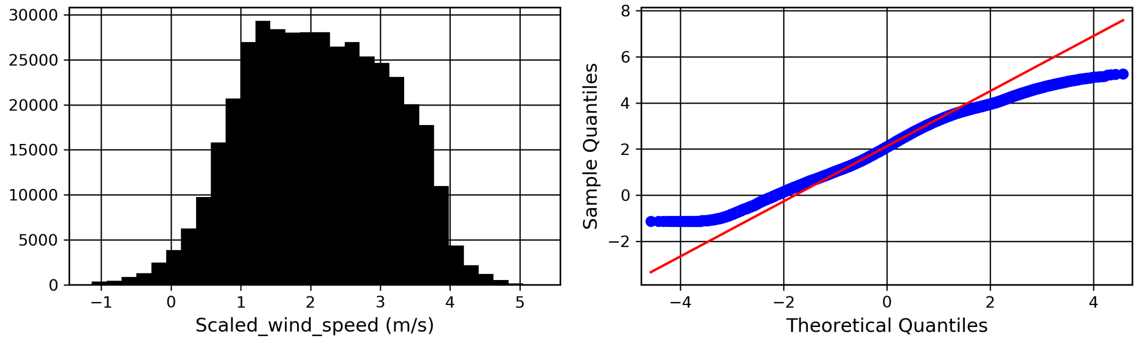
Step 6 is widely used in time series modeling and it is formally referred to as residual diagnostics. Residual is the difference between fitting values and the actual values in the training data. Looking into the statistical properties of residuals can provide intuition about the trained models. In general, the residuals are expected to be random. If there are patterns in the residuals, it might be possible that the model has not captured all the useful information in the training data and thus might be improved. The first step is to look at the residual overtime, which is the time plot for residuals. As mentioned, it is expected that residuals' time plot to be centered around 0 and there is no seasonality or trend-cycle components. Then, the ACF plot provides information about the autocorrelations left in the residuals. If there are still autocorrelations, the model could be improved by tuning the models. There are two properties that are useful but not necessary for residuals to have, which are normal distribution and constant variance. To check if a variable is normally distributed, the quantile plot provides useful visualization for the comparison between a normal and the residual's distribution. However, sometimes the input signal might consist of extreme values or high variations. And this happens to be the case of the wind speed data, as described in 2.2.

This section follows the aforementioned procedures to select the order of each parameter for ARIMAX. The identification of time series components is presented in subsection 2.2.1 and the wind speed series shows strong trend-cycle components but no significant seasonal components. As shown in fig. 3a, the variance is not stable throughout the whole series. In addition, fig. 17a shows that the distribution of wind speed is more like Weibull distributed instead of Gaussian. Thus, Box-Cox transformation is used to stabilize the variance. Box-Cox transformation is essentially a power transformation method that is often used to stabilize the variance and make the data more normal distributed (Box and Cox 1964).

Fig. 17b shows the wind speed after Box-Cox transformation. The next step is to eliminate the trend components by differencing the time series. The differenced wind speed in fig. 18 shows that the trend components are mostly eliminated but the variance of the series still changes over time. However, as discussed in section 2.2, rather than time dependent, the changes of variance are very likely to be influenced by external factors such as the wind farm terrain or the directions of wind speed. In such a case, it is difficult to stabilize the variance if the changes are affected by external variables. Furthermore, running the Augmented Dickey-Fuller test on the differenced wind speed data suggests that the null hypothesis is rejected, which means the differenced wind speed series is not likely to have time-dependent structure (Dickey and Fuller 1979). Since the null hypothesis of the unit root test can already be rejected by differencing the series once, the order d is set to 1 so that the differencing is handled by the hyperparameter d in ARIMAX.



(a) Initial distribution of wind speed



(b) Wind speed distribution after Box-Cox

Figure 17 Box-Cox Transformation

The next step is to select possible orders of p from PACF and q from ACF. Fig. 19a shows little correlations but there is 1 significant spike in ACF plot and around 10 to 16 in PACF plot. These spikes indicate that the upper bounds of (p, q) are $(16, 1)$.

Having the possible orders of each hyperparameter, the most promising model can be selected by calculating the information criteria using grid search on $ARIMAX(p, d, q)$, where $p = \{0, 1, ..., 15, 16\}$, $d = 1$, $q = \{0, 1\}$. The grid search result suggests that the order $(12, 1, 1)$ might be the most promising model. However, the information criterion does not guarantee to choose the best model. The step to choose the upper bounds of the orders p and q is also based on several heuristic rules. Thus, residual diagnostics is still necessary as a final step before the final evaluation to check whether the model can be improved.

Looking into the ACF plot of the residuals fig. 31 in appendix A reveals that the autocorrelations in the signal are mostly captured by. Following the procedures of Box–Jenkins method, it is decided to use $ARIMAX(12, 1, 1)$ for performance evaluation.

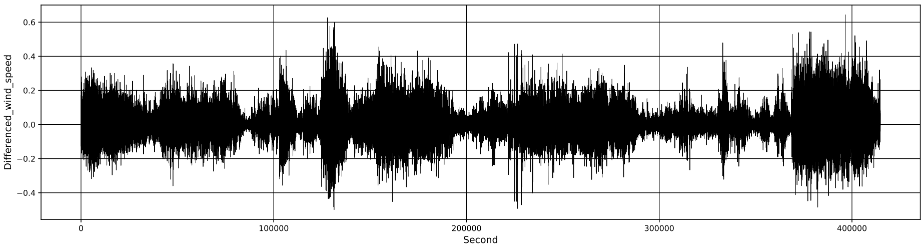
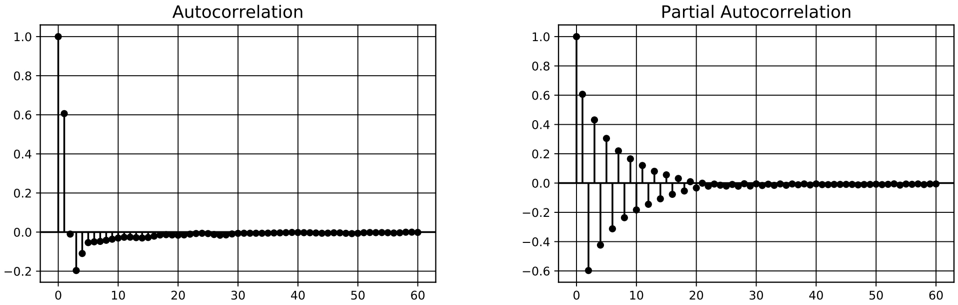


Figure 18 Differenced wind speed



(a) ACF plot **(b)** PACF plot

Figure 19 ACF and PACF

5.3 Long Short-term Memory

Neural network based models have many hyperparameters ranging from structural hyperparameters such as the number of layers and neurons to learning rate, dropout rate, and batch size, etc. Due to the models' complexity, training neural networks is very computationally expensive. It is difficult to try out every combination of the hyperparameters.

A commonly used method for hyperparameter optimization is grid search, which basically tries every combination of a predefined grid. However, as mentioned, training neural networks is an empirical process and often takes a significant amount of time, so efficiency is a very important property of the optimization algorithms. In that case, grid search might not be the best option. Bergstra and Bengio (2012) suggest that random search is more efficient in hyperparameter optimization compared to grid search. They also showed that only a few hyperparameters matter for the performance. Since the influence of each parameter is unknown beforehand, using grid search often ends up spending too many resources on the less important hyperparameters. For the above-mentioned reasons, it is decided to adopt a 2-step strategy. First, a grid search is performed to find better LSTM structures in terms of the number of layers and neurons. A random search is followed to optimize other hyperparameters.

In addition, to obtain a more generalized performance of the model before testing, a time series cross-validation is used in combination with grid search and random search. However, in the case of time series data, cross-validation has to preserve the order of the data. That is, randomly partitioning data into training and validation set should be prevented since it might run the risks of using future data to predict the future. Fig. 20 shows the time series cross-validation setting used in the thesis. The model is trained on 3 different training data and evaluate the performance on the correspondent validation set. The final performance for comparisons is the average performance on the 3 validation set.

The whole process follows the coarse-to-fine strategy. That is, the hyperparameters optimization starts with a coarse grid of parameters and tries to find the influences of each parameter on the performance. After identifying some promising values of the hyperparameters, another finer search is set up around that identified values.

In addition, instead of predefining fixed training epochs for every trial, early stopping is used to reframe from overfitting the training data throughout the hyperparameter optimization process. If early stopping is not used, there would be two possible situations. First, the predefined epochs might be too few for a complex model (e.g. 3-layer LSTM). Second, the number of epochs is sufficient for complex models but it might be more than necessary for a simple model and thus cause overfitting. The problem is that we simply don't know the optimal epochs and the number is different for different structures. Thus, monitoring the training and validation loss by early stopping is considered as a solution.

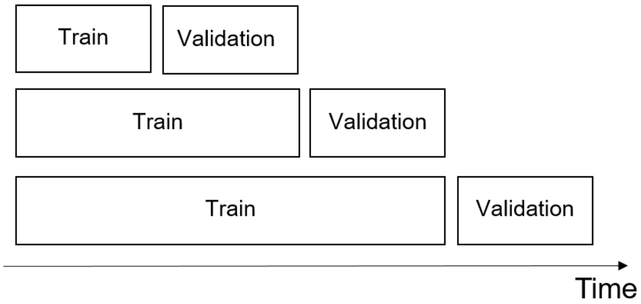


Figure 20 Time series cross validation

5.3.1 Grid Search for LSTM Structure

As mentioned, the first step is to find a better structure for LSTM. A pre-defined structure grid is set up for the first grid search. It is decided to adopt a stacked RNN structure, which means stacking multiple recurrent hidden layers on top of each other. Pascanu et al. (2013) states that by using stacked RNN, the model might be able to use the hidden state at each layer to operate at different timescale. This gives the advantage to model the short-term wind speed time series, which has no clear seasonality patterns but there might exist sophisticated trend or seasonality components that are not easy to spot by human eyes.

The MIMO strategy introduced in 3.1 is applied for LSTM while the other two models adapt the recursive strategy due to the design of the models. As a result, the output data is the next 5-step wind speed while the input is 32 steps of wind speed and wind direction. This is shown in fig. 21, where y_t is the wind speed and x_t is the wind direction at time t. Following the coarse-to-fine strategy, it is decided to try out up to 3-layer stacked LSTM with a pre-defined grid of neurons. Each layer can have 0, 24, 32, or 40 neurons, which results in 39 different structures as shown in fig. 21.

The initial grid search result indicates that 1-layer LSTM performs better. Thus, a second grid search is performed for 1-layer LSTM to find out the most promising number of neurons. The grid contains one parameter, which is the number of neurons $[2, 4, 8, 16, 32]$. Each structure is trained 10 times to get a more general performance measurement. The result suggested that 1-layer LSTM with 8 neurons is the most promising among other structures that have been tried. The experiment results of the 1st and 2nd LSTM structure grid search can be found in tab. 9 and tab. 13 in the appendix B respectively.

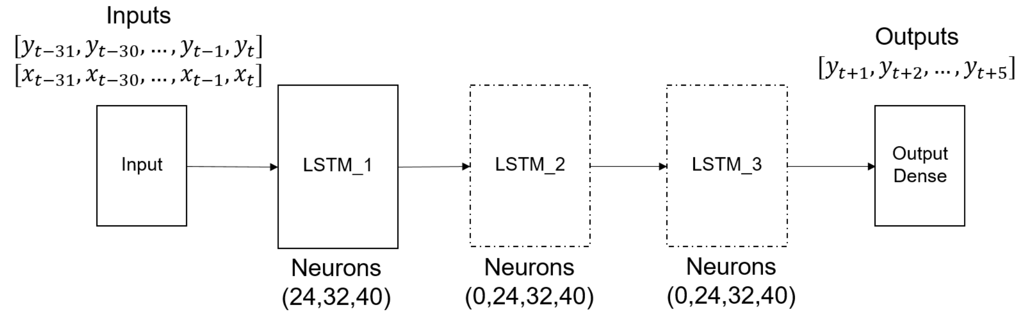

Inputs
$[y_{t-31}, y_{t-30}, \cdots, y_{t-1}, y_t]$
$[x_{t-31}, x_{t-30}, \cdots, x_{t-1}, x_t]$
Input
LSTM_1
LSTM_2
LSTM_3
Outputs
$[y_{t+1}, y_{t+2}, \cdots, y_{t+5}]$
Output Dense
Neurons (24,32,40)
Neurons (0,24,32,40)
Neurons (0,24,32,40)

Figure 21 LSTM structures

5.3.2 Random Search for Learning Rate

Several hyperparameters are considered to be tuned at the beginning such as batch size, learning rate, and dropout rate. Since the grid search suggests that 1-layer is more efficient, applying the dropout rate is out of consideration. The reason is that in an LSTM consisting of one layer, the dropout operation is placed on the input connection, which means some weights in the input layer could be neglected during the backpropagation phase. This is not ideal for a single layer where the dropout is applied for the input weights. The initial experiment also finds that the single LSTM with dropout produce higher validation loss than the ones without. The other finding during the experiments is that training the data with shuffle produces lower validation loss. This can be seen in tab. 11 in the appendix B, the data is obtained by running the same 1st grid search with and without shuffling the training data. In Keras documentation, the default setting of LSTM layer is stateless, which means the state, C_t, is reset for every batch passing through the model (Chollet et al. 2015). It is initially assumed that the order of the input data within the batch has to be kept. However, the empirical results show that shuffling the data helps to improve performance. The order of the data is already preserved by the input sequence, which is set to be 32 lag values of wind speed. In that case, the influence of batch size is expected to be less than shuffling the training data so batch size is also not considered for the random search. In the end, the one hyperparameter left to tune is the learning rate. Random search is used as the next step to finding out promising values for learning rate.

The learning rate scale, s, is used to spread the computation resources evenly among the predefined distribution for the random search. The actual learning rate scaled by s is 10^s. The trials are selected uniformly with s between (-4, -2.5). 64 trials are performed for the learning rate random search.

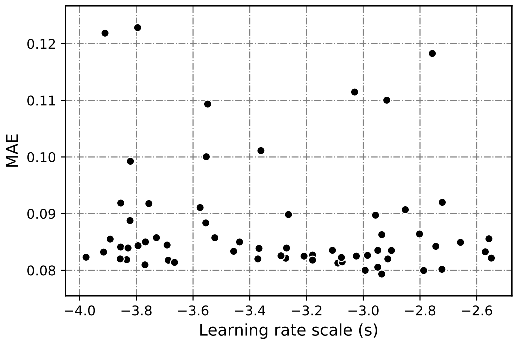


Figure 22 LSTM learning rate random search

Fig. 22 shows the result of the learning rate random search with a scatter plot of learning rate scale against the mean absolute errors. There seem to be no clear relations between MAE and learning rate but we can find that the lowest MAE is obtained around learning rate scale between (-3.0, -2.8). The exact results with the 5 lowest MAE is shown in tab. 4. Finally, it is decided to apply the learning rate scale with the lowest MAE.

Learning rate scale	MAE	MSE
-2.934966883	0.079329475	0.019462823
-2.787109465	0.07995528	0.019529543
-2.993190613	0.079990136	0.019826567
-2.722714339	0.080167791	0.019602741
-2.94892398	0.080535968	0.021161048

Table 4 LSTM learning rate random search

5.3.3 Residual Diagnostics

After the structure and learning rate are decided, residual diagnostics are performed to make sure that the autocorrelations in the original signal are all captured. The time plot on top of fig. 23 indicates that there are no clear trend components in the residuals. The variations of the residuals corresponding to the variation of the original wind speed signal shown in fig. 3. As mentioned in 2.2, the variations are affected by other external factors such as wind farm terrain and are difficult to be captured by the wind speed itself. On the other hand, the ACF plot of the residuals shows that the autocorrelations are mostly captured by LSTM. Although the quantile plot suggests that the residuals are not likely to be normally distributed, this can be explained by the extreme values that exist in the original wind speed signal. Thus, the residual diagnostics suggest that the LSTM model is ready to be evaluated.

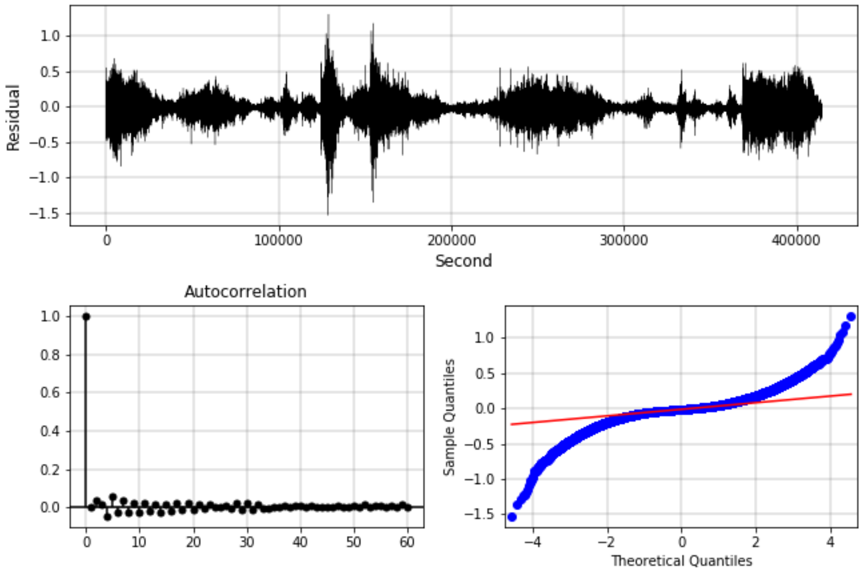


Figure 23 LSTM residual diagnostics

5.4 Adaptive Neural Fuzzy Inference System

Compared to other selected models, the structure of ANFIS is relatively non-flexible. The layers, as well as the output time step, are fixed. The trainable variables are in the 1st and 4th layer. As mentioned in 4.3.2, if the model has input features $x_i, i = 1, 2, 3, ..., n$ and every input has K rules, then there would be $K \times n$ membership functions in the 1st layer in total. Depending on which membership function is used, there might be $2 \times K \times n$ or $3 \times K \times n$ parameters respectively for Gaussian and generalized bell function in this layer. In addition, in the final layer, the resulted number of linear functions would be K^n. And since each linear function has n weights and 1 bias to optimize, the total parameters in the last layer would be $K^n(n + 1)$. This makes further constraints for the number of inputs due to limited computational power. In the end, two combinations of $(inputs, rules)$ are trained for the experiment. First, the wind speed of the previous 16 seconds are used as inputs and each input has 2 rules. The second combination is $(8, 3)$. Both trials applied Gaussian membership function to reduce the trainable parameters.

	Input steps	**Rules**	**RMSE**	**MAE**
ANFIS	16	2	0.975855	1.251997
ANFIS	8	3	0.997170	1.260930

Table 5 ANFIS results

The initial results of ANFIS models in tab. 5 suggest that the prediction error is higher than the baseline models as shown in tab. 6. Due to the unsatisfying performance and the expensive computation, it is decided to shift the focus to the hyperparameters optimization for the other two models.

5.5 WaveNet

As mentioned in 4.3.2, the final softmax activation layer is not used in the thesis. Apart from that, it is decided to follow the WaveNet structure as close as possible. Van Den Oord et al. (2016) uses the kernel width of 2 and the dilated rate of every layer is doubled until a certain point and then repeated. The group of dilated causal layers is referred to as a dilated causal convolution stack in this thesis. For instance, if the model has dilated causal convolution layers of dilated rate $1,2,4,...,64,1,2,4,...,64,1,2,4,...,64$, then the first group of dilated causal convolution layer of dilated rate $1,2,4,...,64$, is referred as the first dilated causal convolution stack.

Due to the design of WaveNet, there are some adjustments for training the model. First, the recursive prediction strategy introduced in 3.1 is used. The design of WaveNet makes it generate a sequence that has the same length as input sequence and shifts one-step ahead. The final step of the predicted sequence is then used recursively to predict until the desired prediction horizon. Secondly, the wind speed is not used as input data for WaveNet. If the wind direction is used for training the model, the model would have to generate a one-step prediction for wind direction as well and use it recursively to predict wind speed. This is not ideal because, during the training phase, the loss function would have to include the prediction of wind direction as well while the prediction performance of interests is the wind speed. The initial experiment comparing WaveNet with/without wind direction as inputs also suggests that including wind direction does not provide clear benefits and even has slightly higher validation loss. The aforementioned experiment result can be found in appendix C tab. 17. Thus, it is decided to use only wind speed as inputs for WaveNet. The input-output pair for each training sample can be seen in fig. 24, where y_t is the wind speed at time t.

The next steps are structure and hyperparameter tuning. The process follows the same procedure as in 5.3. First, a grid search is performed on a predefined grid for the structures. Then, a random search is used to find the learning rate.

5.5.1 Grid Search for WaveNet Structure

Two hyperparameters are directly related to the model, namely the number of layers in the dilated causal convolution stack and the number of stacks. The number of layers in the dilated causal convolution stack influences the number of receptive fields (lag values) since the receptive fields are growing exponentially with the dilated rate, as introduced in 4.3.2. It is decided to try out [4, 5, 6] layers with dilated rate growing exponentially as $1,2,4,...$ in one dilated causal convolution stack. This would correspond to receptive fields of 16, 32, and 64. In addition, the number of stacks, [1, 2, 3, 4, 5], is also defined in the grid search. This would correspondent to 15 possible structures in total. Fig. 24 gives a visualization of the predefined grid for WaveNet structures. There are several dilated causal convolutional stacks and the ones with dashed lines represented optional stacks that are tried out by grid search. Note that fig. 24 is a simplified overview of the model, in reality, there are other components such as skip or residual connections that are not visualized.

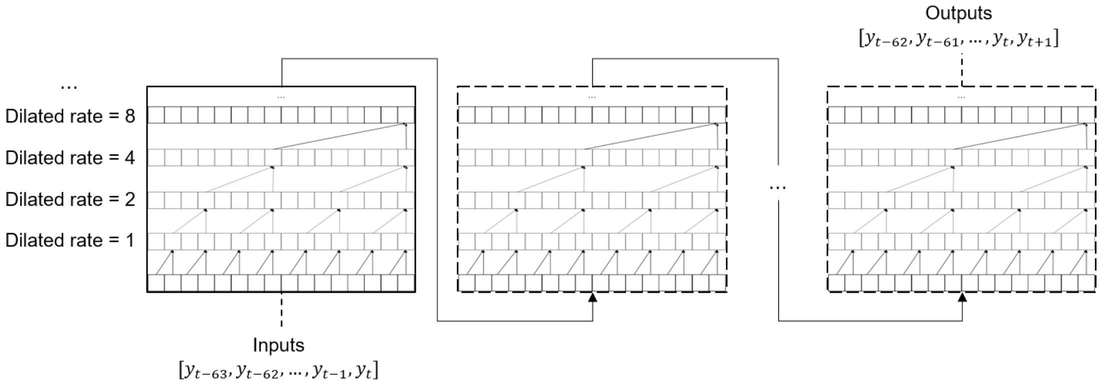

Outputs
$[y_{t-62}, y_{t-61}, \dots, y_t, y_{t+1}]$
...
Dilated rate = 8
Dilated rate = 4
Dilated rate = 2
Dilated rate = 1
...
Inputs
$[y_{t-63}, y_{t-62}, \dots, y_{t-1}, y_t]$

Figure 24 WaveNet structures

The grid search follows the same setting as LSTM, each structure is trained and evaluated on three different validation set. Furthermore, each structure is trained 3 times to get a more general performance. The result indicates that the combination of 6 dilated layers and 4 stacks has the lowest MAE and MSE. The complete results can be found in the appendix C tab. 15.

5.5.2 Random Search for WaveNet Learning Rate

The random search is conducted for learning rate optimization. The other hyperparameters are not considered because it is decided to follow the original WaveNet structure as close as possible. Therefore, the hyperparameter such as dropout rate is not considered. In addition, compared to others, the learning rate is one of the common hyperparameters for optimization.

The learning rate scale, s, is used once again to average the computational resources among the predefined learning rate range, 10^s, where $s \sim U(-4, -2)$. Since the computation is relatively more expensive than the vanilla LSTM, the number of trials for random search is set to 32. Fig. 25 shows the result of random search. Again, there are no clear relations between the learning rate and the validation loss. Therefore, it is decided to use the default setting $s = -3$ to obtain the final trained WaveNet model.

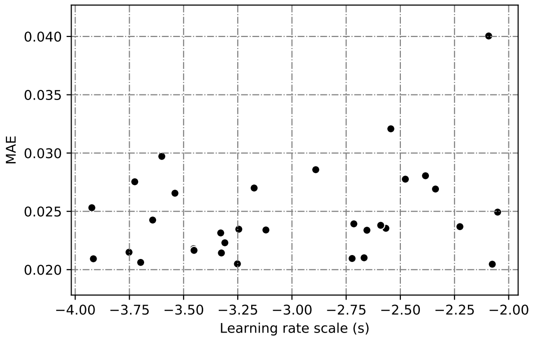


Figure 25 WaveNet learning rate random search

5.5.3 Residual Diagnostics

The residual diagnostics plot for WaveNet in fig. 26 shows similar result as LSTM. The time plot of the residual indicates that the trend-cycle components are captured by the model and the ACF plot for the residuals has no significant spikes. The residual diagnostics suggest that the model is ready for evaluation.

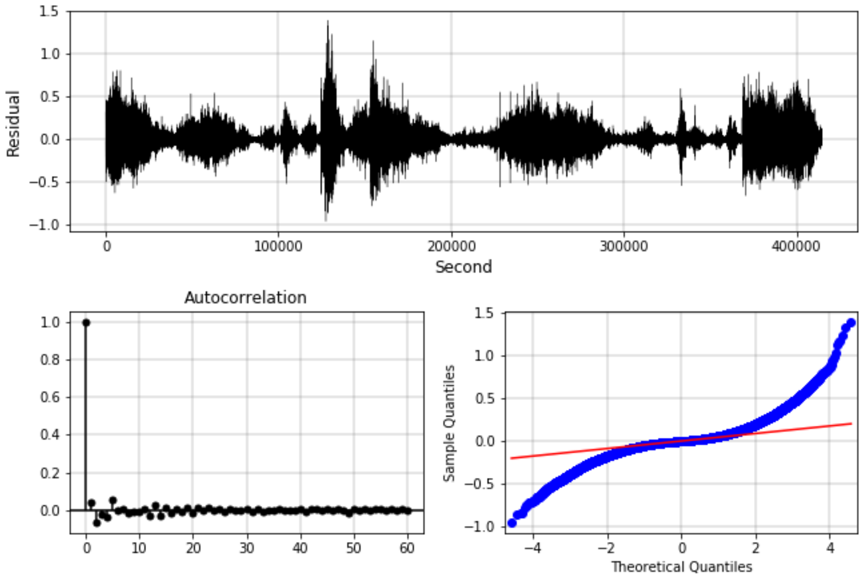


Figure 26 WaveNet residual diagnostics

5.6 Evaluation Metrics

Forecast error is the difference between the actual observed value and prediction. This is similar to residual but forecast error is calculated on the test data. According to the requirements in section 2.1, the models should be evaluated based on their ability to forecast the next 5-second wind speed every 1 second. Thus, for every 1 second in the test data, there is a forecast error of the next 5-second, which can be calculated as

$$e_t = \sum_{h=1}^{5} e_{T+t+h} = \sum_{h=1}^{5} (y_{T+t+h} - \hat{y}_{T+t+h|T}), \tag{5.2}$$

where t is the step in the test set, T is the size of the training data, and h represents the prediction horizon. Since the performance of each model is evaluated on the same time series, there are no issues regarding different scales. Thus, it is decided to use scale-dependent metrics: Mean Absolute Error (MAE) and Root Mean Square Error (RMSE). Both metrics are on the same scale as the evaluated data. MAE is relatively easy to interpret because it is simply the average over the test data of the absolute error. On the other hand, RMSE is the square root of the average squared error. RMSE penalize the large errors because the errors are squared before averaged. The two metrics can be written as

$$MAE = \frac{1}{n} \sum_{t=1}^{n} |e_t|, \tag{5.3}$$

$$RMSE = \sqrt{\frac{1}{n} \sum_{t=1}^{n} (e_t)^2}, \tag{5.4}$$

where n is the test data sample size. The selected models would be evaluated and compared based on these two metrics.

5.7 Results and Discussion

After the structures and hyperparameters are fixed, the models are evaluated with the test data, which are the last 20% of the data.

The overall results comparison is shown in tab. 6 and fig. 27 provides the error comparison step by step for both RMSE and MSE. The performance of ARIMAX is slightly better than the persistence model. However, looking into the error comparison step by step in fig. 27, it can be seen that the first step prediction of ARIMAX is significantly better than the naive persistence model compared to the predictions of other steps.

The results show that the prediction errors can be improved by using the selected machine learning models, especially LSTM and WaveNet. Both models reduce the errors in all 5 prediction steps, while ANFIS is able to improve the error for the first step but start to accumulate errors as the prediction horizon increases. LSTM has the lowest errors and WaveNet is able to obtain similar performance for the first 3 steps but the errors are accumulated and result in larger errors than LSTM in the 4th and 5th steps.

Model	RMSE	MAE
Persistence	0.779194	1.053945
ARIMAX	0.705899	0.939178
LSTM	0.671391	0.870615
ANFIS	0.975855	1.251997
WaveNet	0.683881	0.878439

Table 6 Prediction errors

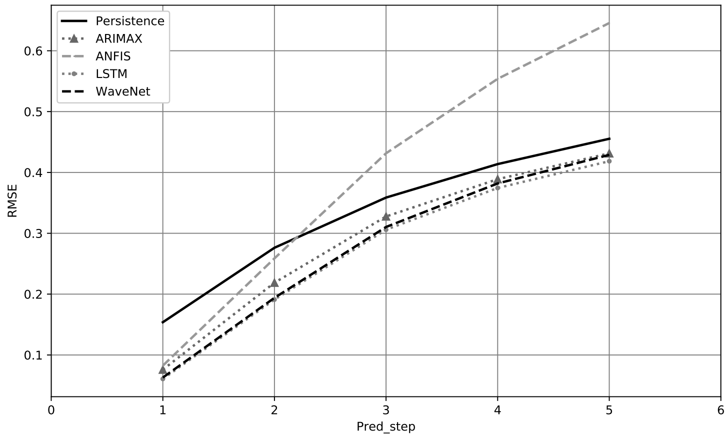
Model	North-East		South-East		South-West		North-West	
	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE
Persistence	0.3303	0.3565	1.0598	1.5043	0.7944	1.1078	0.4695	0.7222
ARIMAX	0.3096	0.3411	0.9517	1.3246	0.7268	0.9902	0.4301	0.6514
LSTM	0.3047	0.3435	0.9020	1.2248	0.6946	0.9152	0.4038	0.5886
ANFIS	0.3181	0.3915	1.0825	1.5674	1.3400	1.6345	0.5245	0.8100
WaveNet	0.3017	0.3132	0.9195	1.2480	0.7112	0.9315	0.4115	0.5991

Table 7 1 step prediction erorrs by directions

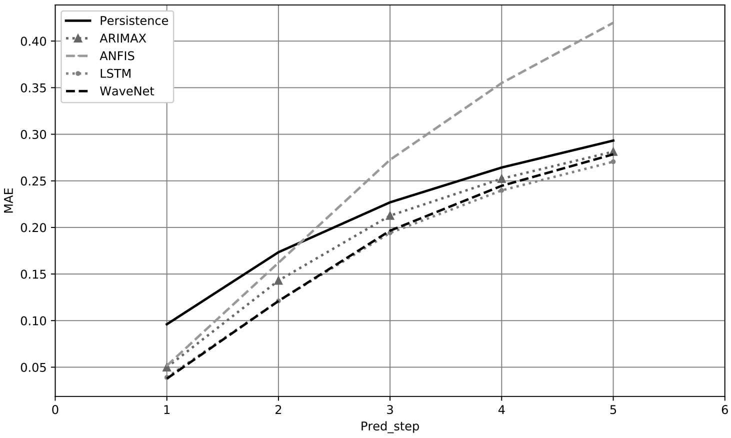
Tab. 7 shows the comparisons of errors categorized by 4 wind directions defined in section 2.2 Note that the wind speed from the south-east (S-E) has the highest prediction error while wind speed from the north-east (N-E) has the lowest error, this correspondent to the variation of wind speed from different directions. As illustrated in fig. 28b, the wind speed from S-E has the highest variant and wind speed from N-E has the lowest in the test data set. The other possible reason for the high error for S-E is that the model is not trained on wind speed from S-E. As fig. 3b shows, there are rarely wind speeds from S-E in the training data.

Despite LSTM and WaveNet reduce prediction errors than persistence in general, it can be seen in fig. 30 that the models still have difficulties capturing the variation of wind speed in the next 5 seconds. The figure shows that most models are able to lower the prediction error for one-second ahead wind speed but the errors increase as the prediction step.

One reason might be the chaotic characteristic of short-term wind speed. As analyzed in section 2.2.1, there exists no dominant periodic components in the wind speed data. The chaotic characteristic makes it difficult for statistical models to capture the patterns of short-term wind speed. Secondly, there might be hidden features that are not available in the given dataset. For example, section 5.2 shows that there are still variances exist



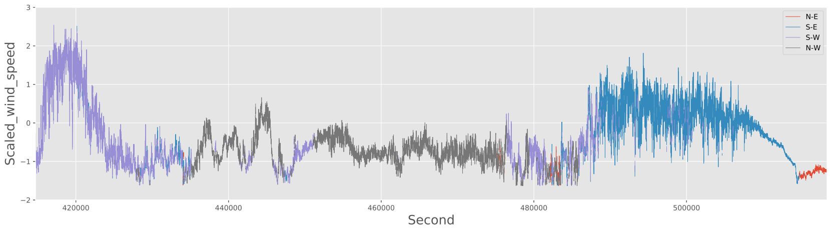
(a) RMSE



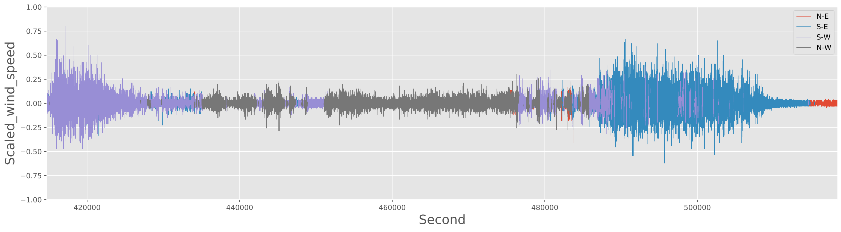
(b) MAE

Figure 27 Step by step results comparison

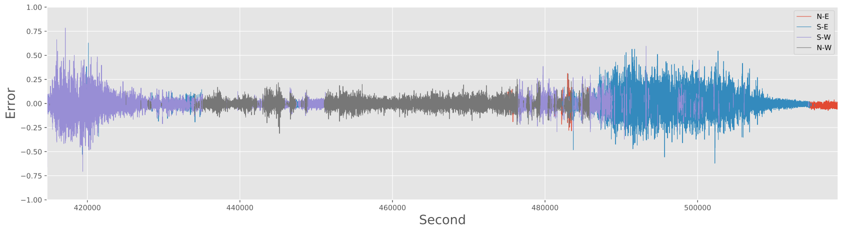
in the series although the 1st order difference has applied and eliminated the trend-cycle components. The Augmented Dickey-Fuller test on the differenced wind speed data suggests that the differenced wind speed series is not likely to have time-dependent structure, which means the remained variances might be affected by other factors such as wind farm terrain. Despite including an additional feature, wind direction, to train the models, the result shows that the implemented models are not able to simulate the variation of wind speed. Therefore, it is suspected that there are hidden features that can be used to improve performance.



(a) Test wind speed

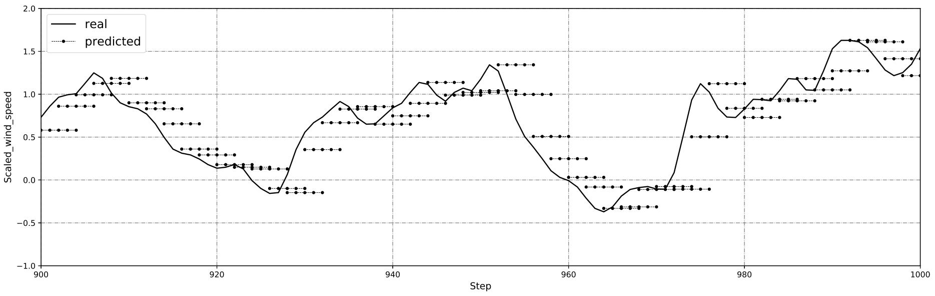


(b) Test differenced wind speed

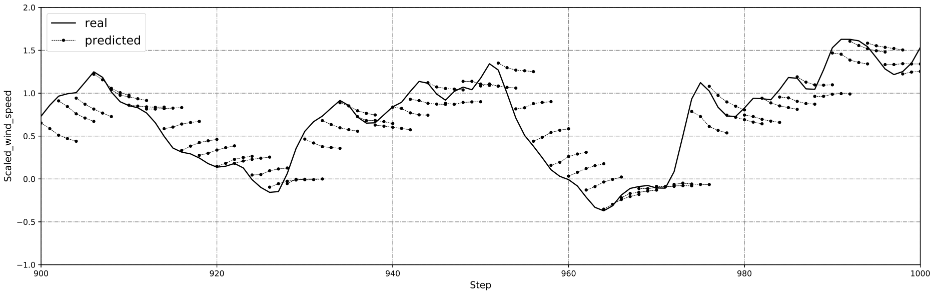


(c) LSTM errors

Figure 28 Timeplot of test wind speed

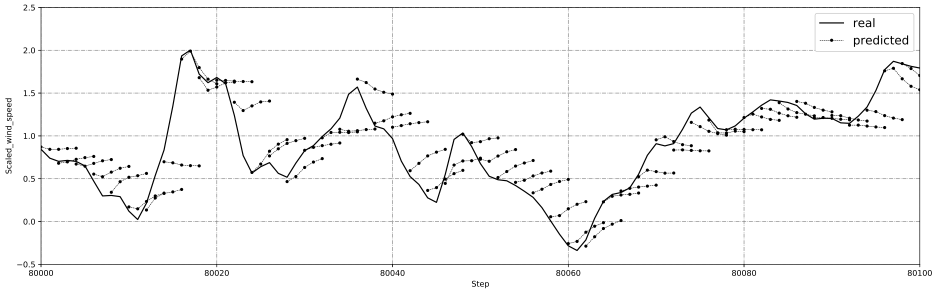


(a) Persistence

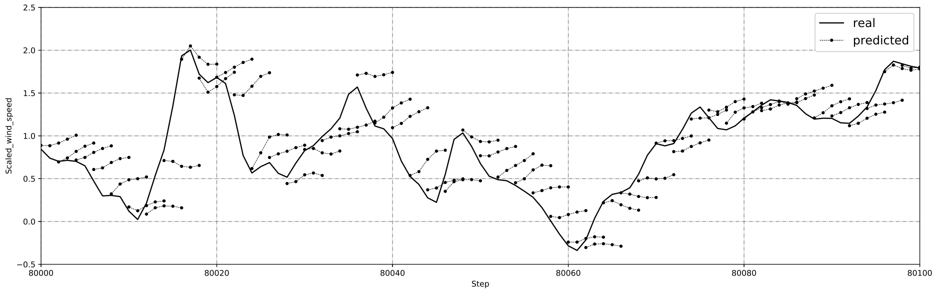


(b) ARIMA

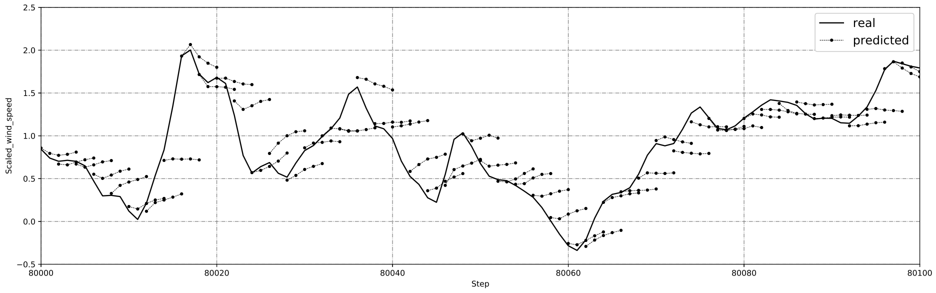
Figure 29 Results of baseline models



(a) LSTM



(b) ANFIS



(c) WaveNet

Figure 30 Results of machine learning models

6 Conclusion and Outlook

The thesis compared the performance of long short-term memory, adaptive fuzzy inference system, and WaveNet for the problem of short-term multi-step ahead wind speed prediction. Both long short-term memory and WaveNet show potential improvements in predicting short-term wind speed. While the result suggests that it is possible to apply the selected models to improve the prediction errors compared to the naive persistence model in general, the models still have difficulties to capture the variation of wind speed with 5-second prediction horizon.

There are a few directions that can be investigated further for future research. Firstly, collecting more wind speed data might be helpful. Although the wind speed data shows chaotic variation in the short-term horizon, it is suspected that once there are more data available for training, the patterns of these short-term chaotic behaviors might be captured by machine learning models. Secondly, while one of the advantages of machine learning models is their ability to reduce preprocessing efforts, it might still be helpful to preprocess the input sequence. Compared to ARIMAX, which applies differencing and Box-Cox transformation, standardization is the only preprocessing used by machine learning models in the thesis. Applying more preprocessing methods might be helpful for these models by reducing the difficulties of the task. In addition, while the evaluation metrics used in this thesis provide the general performance measurement, the true benefits of adopting the selected models still need to be evaluated by the model predictive control system since the objective is to supplement the development of wind speed forecasting model used in the wind turbine control system.

To sum up, the thesis has demonstrated that machine learning models are able to predict short-term wind speed with lower errors than the persistence model. LSTM and WaveNet show their ability to predict one second ahead wind speed but have difficulties capturing the variation of wind speed with sufficient prediction horizon. Several suggestions are proposed above for future research directions.

Appendix

A ARIMAX

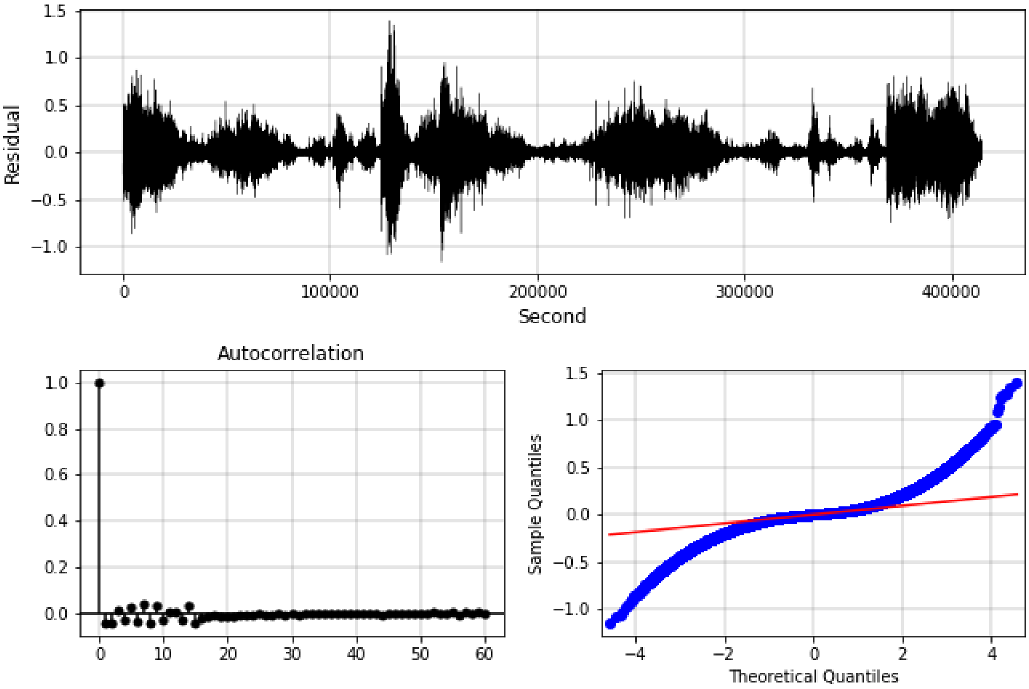


Figure 31 ARIMAX residual diagnostics

B LSTM Hyperparameters Tuning

Neurons	MAE	MSE	MAE rank	MSE rank
32; 0; 0	0.083313983	0.021319492	1	2
24; 0; 0	0.083935145	0.021111014	2	1
40; 0; 0	0.084185255	0.021901126	3	4
40; 40; 0	0.08419166	0.021396151	4	3
24; 32; 0	0.084522383	0.022974546	5	7
40; 24; 32	0.086199971	0.022989031	6	8
40; 32; 24	0.086228299	0.022909838	7	6
32; 40; 32	0.086773412	0.022767596	8	5
32; 24; 0	0.087964543	0.023551507	9	9
32; 40; 24	0.08819471	0.023676379	10	10
24; 32; 32	0.088717089	0.025318701	11	12
40; 24; 24	0.089580256	0.024177011	12	11
24; 24; 24	0.090854982	0.02576106	13	13
32; 24; 40	0.090884611	0.030444899	14	15
32; 40; 0	0.091047697	0.026666664	15	14
24; 40; 32	0.092941294	0.033025711	16	19
32; 32; 24	0.09479118	0.032867352	17	17
24; 40; 24	0.097389939	0.032300407	18	16
40; 32; 32	0.098708169	0.036956752	19	22
40; 32; 40	0.098987824	0.035604471	20	21
24; 24; 40	0.099586295	0.03290765	21	18
40; 40; 24	0.099842241	0.040883993	22	27
32; 32; 0	0.100808745	0.035228928	23	20
32; 40; 40	0.101701651	0.038367471	24	25
40; 24; 40	0.102163021	0.042825558	25	28
32; 32; 32	0.102532084	0.038277235	26	24
24; 24; 32	0.105039077	0.0370716	27	23
24; 32; 24	0.105380226	0.050442916	28	32
24; 32; 40	0.106281545	0.046349973	29	29
40; 24; 0	0.10790362	0.048237587	30	31
24; 40; 0	0.108801586	0.040418866	31	26
32; 24; 32	0.109746569	0.058349553	32	36
24; 40; 40	0.110618325	0.048147584	33	30
24; 24; 0	0.111557476	0.053906795	34	34
40; 40; 40	0.112316825	0.051934963	35	33
32; 24; 24	0.113653457	0.056528125	36	35
40; 40; 32	0.129645621	0.078424604	37	38
40; 32; 0	0.131824967	0.072959935	38	37
32; 32; 40	0.137861615	0.098016516	39	39

Table 9 LSTM 1st structure grid search

Neurons	MAE	MSE	MAE without shuffle	MSE without shuffle
32; 0; 0	0.083313983	0.021319492	0.120135286	0.037435746
24; 0; 0	0.083935145	0.021111014	0.122210923	0.043394542
40; 0; 0	0.084185255	0.021901126	0.116924711	0.037855505
40; 40; 0	0.08419166	0.021396151	0.129348727	0.046396076
24; 32; 0	0.084522383	0.022974546	0.123617393	0.039318345
40; 24; 32	0.086199971	0.022989031	0.132230141	0.047117143
40; 32; 24	0.086228299	0.022909838	0.121733735	0.037943317
32; 40; 32	0.086773412	0.022767596	0.121384519	0.037027162
32; 24; 0	0.087964543	0.023551507	0.138394889	0.052931123
32; 40; 24	0.08819471	0.023676379	0.125878121	0.039846754
24; 32; 32	0.088717089	0.025318701	0.120305648	0.036476944
40; 24; 24	0.089580256	0.024177011	0.124077105	0.040380827
24; 24; 24	0.090854982	0.02576106	0.123565678	0.041383725
32; 24; 40	0.090884611	0.030444899	0.122148628	0.037447657
32; 40; 0	0.091047697	0.026666664	0.133068317	0.057661646
24; 40; 32	0.092941294	0.033025711	0.120935214	0.037800218
32; 32; 24	0.09479118	0.032867352	0.122015546	0.036864086
24; 40; 24	0.097389939	0.032300407	0.119942188	0.035535475
40; 32; 32	0.098708169	0.036956752	0.123914168	0.039555157
40; 32; 40	0.098987824	0.035604471	0.125305996	0.039066688
24; 24; 40	0.099586295	0.03290765	0.120373481	0.036666064
40; 40; 24	0.099842241	0.040883993	0.126551353	0.03994186
32; 32; 0	0.100808745	0.035228928	0.142125859	0.073887124
32; 40; 40	0.101701651	0.038367471	0.122780585	0.038001315
40; 24; 40	0.102163021	0.042825558	0.130469334	0.042187644
32; 32; 32	0.102532084	0.038277235	0.127165542	0.043987239
24; 24; 32	0.105039077	0.0370716	0.122310773	0.037172963
24; 32; 24	0.105380226	0.050442916	0.118848562	0.034713031
24; 32; 40	0.106281545	0.046349973	0.124078549	0.039031008
40; 24; 0	0.10790362	0.048237587	0.117450124	0.038323441
24; 40; 0	0.108801586	0.040418866	0.124412269	0.041331133
32; 24; 32	0.109746569	0.058349553	0.12112665	0.036850166
24; 40; 40	0.110618325	0.048147584	0.12267462	0.037999326
24; 24; 0	0.111557476	0.053906795	0.122836156	0.039287698
40; 40; 40	0.112316825	0.051934963	0.12619523	0.037573936
32; 24; 24	0.113653457	0.056528125	0.122989019	0.037391027
40; 40; 32	0.129645621	0.078424604	0.126680538	0.040119322
40; 32; 0	0.131824967	0.072959935	0.138236087	0.064226276
32; 32; 40	0.137861615	0.098016516	0.125388431	0.037932403

Table 11 LSTM 1st structure grid search with/without shuffle

Neurons	MAE	MSE	MAE rank	MSE rank
8	0.082035692	0.021018582	1	1
32	0.086004021	0.023333708	2	3
16	0.086066334	0.025094605	3	5
4	0.08661901	0.024849302	4	4
2	0.08787593	0.022864723	5	2

Table 13 LSTM 2nd structure grid search

C WaveNet Hyperparameters Tuning

Dilated layers	Stacks	MAE	MSE	MAE rank	MSE rank
6	4	0.02131198	0.001194835	1	1
5	5	0.022404467	0.001300674	2	3
4	2	0.023023411	0.001331122	3	7
4	3	0.023391991	0.001407648	4	10
5	2	0.023418163	0.001388046	5	8
6	3	0.023611885	0.001284141	6	2
6	5	0.023666259	0.001514608	7	14
4	5	0.023722269	0.001309584	8	4
6	2	0.023869234	0.001318755	9	6
6	1	0.024205545	0.001397651	10	9
4	4	0.024264273	0.001314773	11	5
5	1	0.024570826	0.001455353	12	11
5	3	0.024616872	0.001478978	13	13
5	4	0.024891809	0.001478411	14	12
4	1	0.026215726	0.001678942	15	15

Table 15 WaveNet structure grid search

Dilated layers	Stacks	MAE	MSE	MAE with vdir	MSE with vdir
4	1	0.02131198	0.001194835	0.03230178	0.00265148
4	2	0.022404467	0.001300674	0.033549401	0.00368818
4	3	0.023023411	0.001331122	0.023976823	0.001537333
4	4	0.023391991	0.001407648	0.025485393	0.001653119
4	5	0.023418163	0.001388046	0.026341825	0.001594477
5	1	0.023611885	0.001284141	0.026972933	0.001554568
5	2	0.023666259	0.001514608	0.023672418	0.001462597
5	3	0.023722269	0.001309584	0.022489868	0.001234663
5	4	0.023869234	0.001318755	0.024077983	0.001291526
5	5	0.024205545	0.001397651	0.025171957	0.001554961
6	1	0.024264273	0.001314773	0.021227198	0.001197684
6	2	0.024570826	0.001455353	0.026608468	0.001576659
6	3	0.024616872	0.001478978	0.023822689	0.001469852
6	4	0.024891809	0.001478411	0.022256942	0.00120248
6	5	0.026215726	0.001678942	0.022922234	0.001374011

Table 17 WaveNet structure grid search with/without vdir

References

Arlot, S., Celisse, A., et al. 2010. “A survey of cross-validation procedures for model selection,” *Statistics surveys* (4), pp. 40–79.

Bai, S., Kolter, J. Z., and Koltun, V. 2018. “An empirical evaluation of generic convolutional and recurrent networks for sequence modeling,” *arXiv preprint arXiv:180301271* .

Bengio, Y., Simard, P., Frasconi, P., et al. 1994. “Learning long-term dependencies with gradient descent is difficult,” *IEEE transactions on neural networks* (5:2), pp. 157–166.

Bergstra, J., and Bengio, Y. 2012. “Random search for hyper-parameter optimization,” *Journal of Machine Learning Research* (13:Feb), pp. 281–305.

Bishop, C. M. 2006. *Pattern recognition and machine learning*, springer.

Box, G. E., and Cox, D. R. 1964. “An analysis of transformations,” *Journal of the Royal Statistical Society: Series B (Methodological)* (26:2), pp. 211–243.

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

Castellanos, F., and James, N. 2009. “Average hourly wind speed forecasting with ANFIS,” in *11th Americas Conf. on Wind Eng.(11 ACWE).*

Chiu, J. P., and Nichols, E. 2016. “Named entity recognition with bidirectional LSTM-CNNs,” *Transactions of the Association for Computational Linguistics* (4), pp. 357–370.

Chollet, F., et al. 2015. “Keras,” https://github.com/fchollet/keras.

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

Foley, A. M., Leahy, P. G., Marvuglia, A., and McKeogh, E. J. 2012. “Current methods and advances in forecasting of wind power generation,” *Renewable Energy* (37:1), pp. 1–8.

Gehring, J., Auli, M., Grangier, D., Yarats, D., and Dauphin, Y. N. 2017. “Convolutional sequence to sequence learning,” in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, JMLR. org, pp. 1243–1252.

Géron, A. 2017. *Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems*, " O'Reilly Media, Inc.".

Giebel, G., Brownsword, R., Kariniotakis, G., Denhard, M., and Draxl, C. 2011. "The state-of-the-art in short-term prediction of wind power: A literature overview," *ANEMOS plus* .

Goodfellow, I., Bengio, Y., and Courville, A. 2016. *Deep learning*, MIT press.

Gosk, A. 2011. *Model predictive control of a wind turbine*, Ph.D. thesis, Master's thesis, Technical University of Denmark.

Graves, A., Mohamed, A.-r., and Hinton, G. 2013. "Speech recognition with deep recurrent neural networks," in *2013 IEEE international conference on acoustics, speech and signal processing*, IEEE, pp. 6645–6649.

He, K., Zhang, X., Ren, S., and Sun, J. 2016. "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778.

Henriksen, L. C. 2010. *Model predictive control of wind turbines*, DTU Informatics.

Hochreiter, S. 1998. "The vanishing gradient problem during learning recurrent neural nets and problem solutions," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* (6:02), pp. 107–116.

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

Huang, C.-J., and Kuo, P.-H. 2018. "A short-term wind speed forecasting model by using artificial neural networks with stochastic optimization for renewable energy systems," *Energies* (11:10), p. 2777.

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

Jain, A. K., Mao, J., and Mohiuddin, K. 1996. "Artificial neural networks: A tutorial," *Computer* (:3), pp. 31–44.

Jang, J.-S. 1993. "ANFIS: adaptive-network-based fuzzy inference system," *IEEE transactions on systems, man, and cybernetics* (23:3), pp. 665–685.

Jung, J., and Broadwater, R. P. 2014. "Current status and future advances for wind speed and power forecasting," *Renewable and Sustainable Energy Reviews* (31), pp. 762–777.

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

Kulkarni, P. A., Dhoble, A. S., and Padole, P. M. 2019. “Deep neural network-based wind speed forecasting and fatigue analysis of a large composite wind turbine blade,” *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science* (233:8), pp. 2794–2812.

LeCun, Y., Bottou, L., Bengio, Y., Haffner, P., et al. 1998. “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE* (86:11), pp. 2278–2324.

Luong, M.-T., Sutskever, I., Le, Q. V., Vinyals, O., and Zaremba, W. 2014. “Addressing the rare word problem in neural machine translation,” *arXiv preprint arXiv:14108206* .

McCulloch, W. S., and Pitts, W. 1943. “A logical calculus of the ideas immanent in nervous activity,” *The bulletin of mathematical biophysics* (5:4), pp. 115–133.

Nielsen, M. A. 2015. *Neural networks and deep learning*, vol. 25, Determination press San Francisco, CA, USA:.

Olah, C. 2015. “Understanding LSTM Networks,” .

Oore, S., Simon, I., Dieleman, S., and Eck, D. 2017. “Learning to create piano performances,” in *NIPS Workshop on Machine Learning for Creativity and Design. https://nips2017creativity. github. io/doc/Learning_Piano. pdf*.

Orhan, A. E., and Pitkow, X. 2017. “Skip connections eliminate singularities,” *arXiv preprint arXiv:170109175* .

Pascanu, R., Gulcehre, C., Cho, K., and Bengio, Y. 2013. “How to construct deep recurrent neural networks,” *arXiv preprint arXiv:13126026* .

Potter, C. W., and Negnevitsky, M. 2006. “Very short-term wind forecasting for Tasmanian power generation,” *IEEE Transactions on Power Systems* (21:2), pp. 965–972.

Qin, Q., Lai, X., and Zou, J. 2019. “Direct Multistep Wind Speed Forecasting Using LSTM Neural Network Combining EEMD and Fuzzy Entropy,” *Applied Sciences* (9:1), p. 126.

Rumelhart, D. E., Hinton, G. E., Williams, R. J., et al. 1988. “Learning representations by back-propagating errors,” *Cognitive modeling* (5:3), p. 1.

Russell, S. J., and Norvig, P. 2016. *Artificial intelligence: a modern approach*, Malaysia; Pearson Education Limited,.

Shamshirband, S., Petković, D., Anuar, N. B., Kiah, M. L. M., Akib, S., Gani, A., Ćojbašić, Ž., and Nikolić, V. 2014. “Sensorless estimation of wind speed by adaptive neuro-fuzzy methodology,” *International Journal of Electrical Power & Energy Systems* (62), pp. 490–495.

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

Simonyan, K., and Zisserman, A. 2014. “Very deep convolutional networks for large-scale image recognition,” *arXiv preprint arXiv:14091556* .

Smith, D. A., and Harris, M. 2007. “Wind turbine control having a lidar wind speed measurement apparatus,” US Patent 7,281,891.

Soman, S. S., Zareipour, H., Malik, O., and Mandal, P. 2010. “A review of wind power and wind speed forecasting methods with different time horizons,” in *North American Power Symposium 2010*, IEEE, pp. 1–8.

Takagi, T., and Sugeno, M. 1993. “Fuzzy identification of systems and its applications to modeling and control,” in *Readings in Fuzzy Sets for Intelligent Systems*, Elsevier, pp. 387–403.

Tang, D., Qin, B., and Liu, T. 2015. “Document modeling with gated recurrent neural network for sentiment classification,” in *Proceedings of the 2015 conference on empirical methods in natural language processing*, pp. 1422–1432.

Tascikaraoglu, A., and Uzunoglu, M. 2014. “A review of combined approaches for prediction of short-term wind speed and power,” *Renewable and Sustainable Energy Reviews* (34), pp. 243–254.

Van Den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., and Kavukcuoglu, K. 2016. “Wavenet: A generative model for raw audio,” *CoRR abs/160903499* .

Wang, J., Song, Y., Liu, F., and Hou, R. 2016. “Analysis and application of forecasting models in wind power integration: A review of multi-step-ahead wind speed forecasting models,” *Renewable and Sustainable Energy Reviews* (60), pp. 960–981.

Wang, X., Smith, K., and Hyndman, R. 2006. “Characteristic-based clustering for time series data,” *Data mining and knowledge Discovery* (13:3), pp. 335–364.

WindEurope 2018. “Wind in Power: 2017 European Statistics,” .

Wu, W., Chen, K., Qiao, Y., and Lu, Z. 2016. “Probabilistic short-term wind power forecasting based on deep neural networks,” in *2016 International Conference on Probabilistic Methods Applied to Power Systems (PMAPS)*, IEEE, pp. 1–8.

Zaytar, M. A., and El Amrani, C. 2016. “Sequence to sequence weather forecasting with long short-term memory recurrent neural networks,” *International Journal of Computer Applications* (143:11), pp. 7–11.

Declaration of Authorship

I hereby declare that, to the best of my knowledge and belief, this Master Thesis titled "Comparison of Machine Learning Methods for Short-term Wind Speed Forecasting" is my own work. I confirm that each significant contribution to and quotation in this thesis that originates from the work or works of others is indicated by proper use of citation and references.

Gifhorn, 2nd July 2019

Tsunghao Huang

Consent Form

for the use of plagiarism detection software to check my thesis

Last name: Huang **First name**: Tsunghao
Student number: 436499 **Course of study**: Information Systems
Address: Fischerweg 58, 38518 Gifhorn
Title of the thesis: "Comparison of Machine Learning Methods for Short-term Wind Speed Forecasting"

What is plagiarism? Plagiarism is defined as submitting someone else's work or ideas as your own without a complete indication of the source. It is hereby irrelevant whether the work of others is copied word by word without acknowledgment of the source, text structures (e.g. line of argumentation or outline) are borrowed or texts are translated from a foreign language.

Use of plagiarism detection software The examination office uses plagiarism software to check each submitted bachelor and master thesis for plagiarism. For that purpose the thesis is electronically forwarded to a software service provider where the software checks for potential matches between the submitted work and work from other sources. For future comparisons with other theses, your thesis will be permanently stored in a database. Only the School of Business and Economics of the University of Münster is allowed to access your stored thesis. The student agrees that his or her thesis may be stored and reproduced only for the purpose of plagiarism assessment. The first examiner of the thesis will be advised on the outcome of the plagiarism assessment.

Sanctions Each case of plagiarism constitutes an attempt to deceive in terms of the examination regulations and will lead to the thesis being graded as "failed". This will be communicated to the examination office where your case will be documented. In the event of a serious case of deception the examinee can be generally excluded from any further examination. This can lead to the exmatriculation of the student. Even after completion of the examination procedure and graduation from university, plagiarism can result in a withdrawal of the awarded academic degree.

I confirm that I have read and understood the information in this document. I agree to the outlined procedure for plagiarism assessment and potential sanctioning.

Gifhorn, 2$^{\text{nd}}$ July 2019

Tsunghao Huang