Testing a forecasting system for the measuring sites Hannover-Herrenhausen and Ruthe based on neural networks

Bachelor thesis
27. August 2023

Alexander Steding

Bachelor's degree course: B. Sc. Meteorologie
Enrolment number: 10028034
Examiner: Prof. Dr. Björn Maronga
Supervisor: M. Sc. Pierre Monteyne

Abstract

Short term forecasts of meteorological parameters play an important role in many societal processes. Until recently, seasonal autoregressive integrated moving average models (SARIMA) have been used to make forecasts on meteorological time series data. This thesis deploys and evaluates three different neural network forecasting systems, based on long short term memory (LSTM) networks. One univariate LSTM model, one multivariate LSTM model that receives all input parameters, and one multivariate LSTM model that only received correlating inputs. Each forecasting system uses twelve different LSTM submodels to forecast the meteorological parameters at the measuring site, Hannover-Herrenhausen. The forecasting systems are compared with the SARIMA approach and a simple seasonal naive as a baseline model. For the comparison, the root mean squared error and mean absolute scaled error were computed. The neural network based forecasting systems outperform the SARIMA model in every parameter, except precipitation. Using only correlating inputs improved just selected parameter performance. Notably, the optimal window size was analysed to be 24 hours for the networks. The test on a second dataset from the measuring site in Ruthe revealed that the neural forecasting systems possess the ability to generalize on unknown data.

Table of contents

1 Introduction

Short-term forecasting meteorological parameters can be of particular relevance in a magnitude of different aspects of society. Street safety, renewable energy production, and the aviation industry are but a few examples. For forecasting meteorological time series at singular stations, statistical methods like Seasonal Autoregressive Integrated Moving Average models (SARIMA) or Vector Autoregression (VAR) models have been used instead of complex numerical weather prediction methods (Wilks 2019). However, these SARIMA models suffer from increased resource usage on larger forecast horizons during the forecast production, because of complex long-term relations (Tran et al. 2021). Due to progress made in theoretical informatics and graphical processing units, machine learning is on the rise and neural networks have emerged as an alternative solution in time series forecasting. Today, neural networks are not only powering impressive chatbots like ChatGPT through natural language processing (Chowdhary 2020), but also in meteorological predictions. Notably, these neural networks have been successfully deployed to predict air temperature (Eide et al. 2021, Eide et al. 2022, Tran et al. 2021). Over the years, different neural network architectures were proposed for forecasting air temperature in varying situations. Kreuzer, Munz, and Schlüter 2020 combined two-dimensional convolutional neural networks with long short term memory networks to model complex interparametrical dependencies. Eide et al. 2022 suggested a novel method to use spatiotemporal input data from different locations in tower networks. Alerskans et al. 2022 used the novel attention mechanism in complex decoder encoder transformer networks. As shown by Haque, Tabassum, and Hossain 2021 recurrent long short term memory models provide a good balance between model performance and computational efficiency and were therefore chosen as a model architecture.

This thesis tries to implement a series of long short term memory neural networks to forecast all the parameters listed in Table 1 which are measured at the measuring site Hannover-Herrenhausen. The model performance shall be evaluated in comparison to the SARIMA model as a traditional statistical method as well as a seasonal naive baseline model. To evaluate the absolute model performance, the root mean squared error (RMSE) will be used and for the relative improvement over the seasonal naive model, the mean scaled error (MASE). Additionally, this thesis will inspect the importance of parameters and different forecast horizons, as well as input data lengths used in the prediction procedure. Finally, the models will be tested on the data for the measuring site Ruthe to check the model's robustness.

variable	abbreviation
air temperature	T_{air}
relative humidity	RH
reduced air pressure	p_{sl}
wind speed 10 m	u_{10}
wind speed 50 m	u_{50}
gust speed 10 m	g_{10}
gust speed 50 m	g_{50}
wind direction u component	φ_u
wind direction v component	φ_v
precipitation	P
global radiation	G
diffuse radiation	D

Table 1: Target parameters for the neural networks. Each parameter receives its network.

The remainder of this thesis is structured as follows: First Section 2 introduces the theoretical foundations required for the thesis. Section 3 provides insight into the different model architectures and the data used for training. Finally, in Section 4 are the results of the evaluation presented and discussed. The code for this thesis can be found at https://codeberg.org/Kleeritter/neuralcast.git.

2 Theoretical foundations

This chapter lays the theoretical foundation for concepts discussed throughout this thesis.

2.1 Statistical models

Different statistical models and methods have been developed to forecast time series data (Wilks 2019). One of those is the SARIMA model. This model was first described by Box, Jenkins, and Reinsel 1970. This model can be broken down into their name-giving submodules: S, AR, I, MA (Korstanje 2021). What all these submodules require is a stationarity in time series. In a stationary time series, the characteristics e.g. mean and variance are the same for slices of the same length (Wilks 2019).

The first submodule is the Autoregression (AR) term. Forecasting the next value of a series X_t in time only depends on the previous value $X_{\text{t-1}}$ and a random variable ε_t, representing the change between the values (Wilks 2019). The autoregression can not only include the previous value $X_{\text{t-1}}$ for forecasting but also the last p values (Wilks 2019). This leads to the general formula for AR processes with order p (Box, Jenkins, and Reinsel 1970, Whittle 1951):

$$X_t = \sum_{i=1}^{p} \varphi_i X_{\text{t-i}} + \varepsilon_t \tag{1}$$

Where
- φ_i : model parameter
- ε_t : prediction error
- p : number of lags

Since forecasted values are calculated by previous values, which in turn are also calculated by previous values, shocks to the time e.g. outlier can propagate through the forecasting. Although this effect decreases with time, autoregressive models can be considered long-memory models (Wilks 2019).

Contrary, the Moving Average (MA) model does not depend on the previous values of the time series, but rather on the errors of the past forecast (Korstanje 2021). For the first value in the series, the average μ of the series is used as a forecast, being the namesake of this model. Analog to the previous formula structure but for the last q errors, the following function can be used (Whittle 1951, Box, Jenkins, and Reinsel 1970):

$$X_t = \mu + \sum_{i=1}^{q} \theta_i \varepsilon_{\text{t-i}} + \varepsilon_t \tag{2}$$

Where
- μ : mean of series X
- ε_t : prediction Error
- θ_i : model parameter

These two submodules can then be combined into the Autoregressive Moving Average (ARMA) model (Whittle 1951):

$$X_t = \mu + \sum_{i=1}^{p} \varphi_i X_{\text{t-i}} + \sum_{i=1}^{q} \theta_i \varepsilon_{\text{t-i}} + \varepsilon_t \tag{3}$$

Thus, creating a model that uses past values and errors of past forecasts to forecast X_t (Korstanje 2021).

Just like the AR and MA submodule, the requirement for a stationary series remains in the ARMA submodule. Often meteorological variables, especially temperature, show strong daily and annual seasonality and are therefore not stationary (Wilks 2019, Desolte and Tippet 2022). To circumvent this problem in time series data, seasonal-differencing can be applied by subtracting the value one season ago $X_{\text{t-s}}$ from the current value X_t (Desolte and Tippet 2022).

$$X'_t = X_t - X_{\text{t-s}} \tag{4}$$

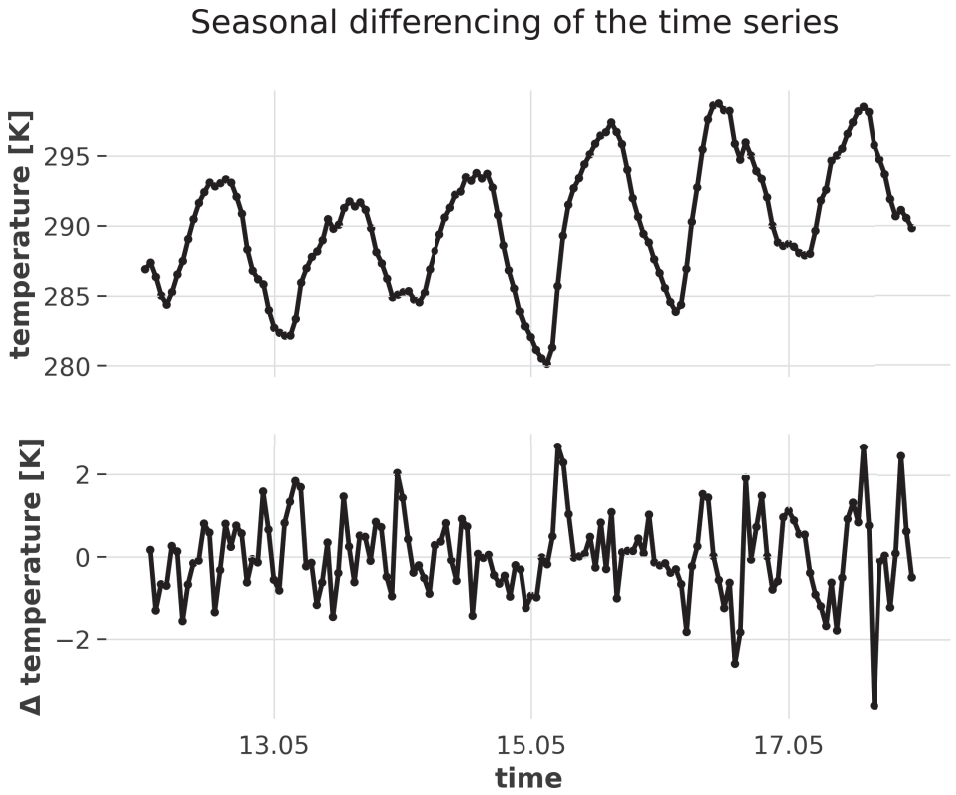


Figure 1: Example of removing daily seasonality by two times seasonal differencing. The first graph shows temperature data with strong seasonality. The second graph shows the two times de-seasonalized data, with first $s = 1$ hour and then $s_2 = 24$ hours.

This de-seasonalized data can then be used inside the ARMA submodule. The resulting two-step model is then called the autoregressive-integrated-moving-average model and marks another submodule, ARIMA (Box et al. 2015). Since the order of the Autoregression process p, the order of the Moving Average process q and the number of differencing needed to reach stationarity d drastically changes the model performance, it is common to list those parameters: ARIMA(p, d, q) (Korstanje 2021, Wilks 2019). For example, ARIMA$(0, 0, 1)$ would be a simple MA process with an AR window of one.

To further improve the ARIMA model, seasonality in the modelling was reintroduced by the Seasonal Autoregressive Integrated Moving Average model (SARIMA) (Box et al. 2015), which just adds another AR and MA process for a given seasonal length s (Box et al. 2015):

$$\begin{aligned} X_t = \mu &+ \sum_{i=1}^{p} \varphi_i X_{\text{t-i}} + \sum_{i=1}^{q} \theta_i \varepsilon_{\text{t-i}} \\ &+ \sum_{i=1}^{P} \Phi_i X_{\text{ts-i}} \sum_{i=1}^{q} \Theta_i \varepsilon_{\text{ts-i}} + \varepsilon_t \end{aligned} \tag{5}$$

Where
- Φ_i, Θ_i : model parameter
- ε_{ts} : prediction error of the seasonal component
- X_{ts} : value of the seasonal component

The model parameters of the seasonal processes Φ_i and Θ_i work as parameters analogous to their non-seasonal counterparts φ_i and θ_i. Just like ARIMA models, SARIMA models are often encountered in their parameter form $\text{SARIMA } (p,d,q) \times (P,D,Q)s$. Where p, d, q are the non-seasonal ARIMA parameters and P, D, Q are the seasonal parameters with the seasonal period s (Korstanje 2021).

As stated above, the choice of the right parameters strongly influences the forecasted values, even more so for SARIMA. It therefore needs a designated strategy described in Section 3.2.2. Nonetheless, SARIMA models have been used to successfully forecast various time series problems, e.g. aeroplane passengers, (Xu, Chan, and T. Zhang 2019), air pollution (Samal et al. 2019) and wind turbine energy production (Tena Garcia et al. 2019).

2.2 Neural networks

Although delivering promising results in forecasting, SARIMA models lack the ability to capture multiple seasonalities since the length of the seasonal period s needs to be set individually (Korstanje 2021). Furthermore, increasing complexity in the time series leads to serious performance hits. Neural artificial networks try to solve some of these problems (Khashei, Bijari, and Hejazi 2012).

The first practical neural networks were described by Rosenblatt 1958. These took inspiration from the signal processing of the human brain. An external stimulus creates an electrical signal that charges a neuron over its dendrites. If the charge exceeds a certain threshold, a so-called action potential activates. This sudden electrical impulse can then, over the synapses of the neurons, activate other connected neurons, et cetera. Each neuron has a different threshold level and different connection strengths to other neurons. (Hudspeth et al. 2013)

With about 84 billion interconnected neuron cells, the human brain is capable of computing complex tasks like image recognition (Von Bartheld, Bahney, and Herculano-Houzel 2016). In an artificial neural network, this basic concept is mimicked. A node representing a neuron gets multiple input values X_n. These values are then adjusted with a weight w_n, representing the connection strength to each input cell, and a bias, which in turn represents the charge threshold of the biological neuron. This charge threshold/bias acts as a barrier, to filter out values that could also be attributed to noise. If the weighted sum of the input values exceeds the bias, the artificial neuron activates and 'fires' to connected neurons (Nielsen 2015).

This gives the basic structure of a so-called perceptron with the output formula (Nielsen 2015):

$$\text{output} = \begin{cases} 0 \text{ if} \sum_n w_n X_n + b \leq 0 \\ 1 \text{ if} \sum_n w_n X_n + b > 0 \end{cases} \tag{6}$$

Where
- w_n : weight for input n
- X_n : input n
- b : bias for the neuron

One perceptron is often not enough to handle complex problems, so multiple perceptrons are stacked in a structure commonly referred to as layers (Nielsen 2015). Each perceptron in the layer receives the same input values but applies different weights to the values and can have different biases. This results in different output values from the individual perceptrons in the layer. Each output value from a layer

can be then used as an input to a different layer. This produces the horizontal netlike structure seen in Figure 2.

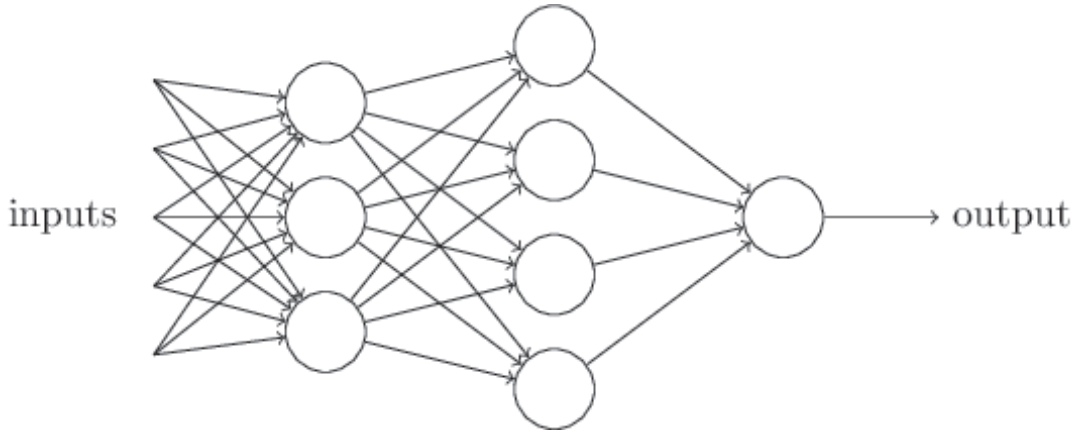


Figure 2: A simple neural network (Nielsen 2015). The circles signify the individual neurons, each row of circles one layer.

Each weight and bias acts as a little turning wheel, a variable, which can be used to create the desired outputs. This presupposes that small changes in the weight and bias lead to small changes in the outcome. However, using the above-mentioned formula to calculate the output, small changes in w_n or b can lead to diametrical different outputs and as discussed later on to a problem called exploding gradient. Therefore, the use of a scaling function, also called the activation function, is necessary for proper training. These functions allow inputs varying from 0 to 1 and not only either 0 or 1 (Sharma, Sharma, and Athaiya 2017).

Over the years different functions like the rectified linear unit, ReLU, have come up in literature (Sharma, Sharma, and Athaiya 2017). Two other types which will be used in this thesis are the sigmoid and hyperbolic tangents function (Sharma, Sharma, and Athaiya 2017). With the sigmoid being defined by

$$\sigma_g(x) = \frac{1}{1 + e^{-x}} \tag{7}$$

and the hyperbolic tangents (tanh) :

$$\sigma_h(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}. \tag{8}$$

This was necessary to enable machine learning for this kind of network. A method often used in training neural networks is the so-called backpropagation algorithm, which was first described by Linnainmaa 1976.

The overall goal of the algorithm is to reduce the error of the final output of the neural network in comparison to the actual value by making small changes to the weights and biases of the neurons. The error may be calculated by using a loss function, e.g. the mean squared error (MSE) (Nielsen 2015).

$$\text{MSE} = \frac{1}{m} \sum_{\text{i}=1}^{m} \left(\hat{y}_i - y_i \right)^2 \tag{9}$$

Where

- y_i : target value
- $\hat{y}_i$: predicted value

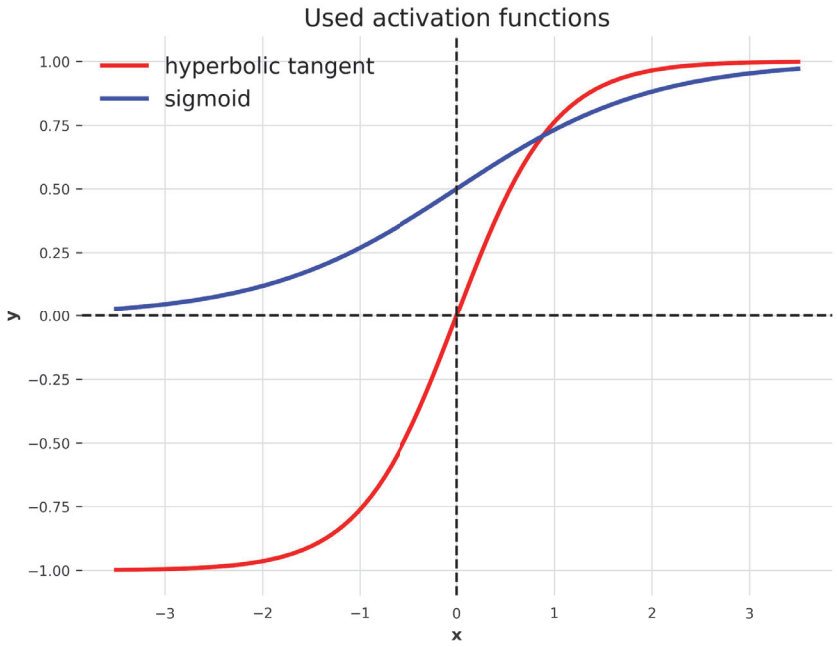


Figure 3: Graphical visualisation of the tanh and sigmoid activaition functions.

The backpropagation algorithm as described in Rumelhart, Hinton, and Williams 1986 goes backwards through the network. Starting with the last layer, the algorithm tries to compute those weights and biases which lead to the smallest error in the loss function. The algorithm then propagates one layer backwards and computes the same operations once more. This continues until the algorithm has reached the first layer of the network (Rumelhart, Hinton, and Williams 1986).

There exist several approaches to how the backpropagation algorithm finds the lowest error inducing weights and biases (Ruder 2016). One of those is Adam, which stands for adaptive moment estimation (Kingma and Ba 2014). Adam is an optimizer based on stochastic gradient descent (SGD), which itself is based on the more primitive gradient descent described in Bottou 1998.

The gradient descent tries to iteratively find a local minimum of a function. After a random starting point, the algorithm calculates the gradient $\nabla_w Q(z, w_t)$ for this weight w_t and subtracts it from the previous weight (Bottou 2012). The speed and success of the algorithm strongly depend on the learning rate η. A large value of η leads to fast results, but it may skip over the minimum. On the other hand, smaller values of η can significantly slow down the algorithm (Bottou 2012) and this can be expressed by:

$$w_{\mathrm{n+1}} = w_t - \eta \frac{1}{n} \sum_{\mathrm{i=1}}^{n} \nabla_w Q(z_i, w_t) \tag{10}$$

Where

- η : learning rate
- w_t : weight at step t
- n : number of samples in the dataset
- $Q(z_i, w_t)$: loss of z_i with weight w_t

After repeating these steps, the algorithm eventually converges to a local minimum (Bottou 2012). For a three-dimensional function, this can be imagined, as a hiker starting at a point on the mountain and

then always taking a step in the direction with the highest slope until he reaches a point where the slope is 0.

Since this algorithm computes the average over all the losses of the dataset, it slows down with larger datasets. The stochastic gradient descent tries to solve this problem by approximating the gradient descent stochastically. Instead of calculating the gradient for every data point in the dataset, it randomly chooses a subset of the data for calculation and updates the parameters afterwards. (Bottou 2012)

$$w_{\mathrm{n+1}} = w_t - \eta \nabla_w Q(z_t, w_t) \tag{11}$$

Where

- z_t : randomly selected subset

These random subsets are also called batches (Nielsen 2015). The size and number of batches are defined beforehand, and the algorithm goes randomly through all available batches. After all batches have been used, an epoch has been completed and the pool of available batches for the algorithm resets (Nielsen 2015).

Further, optimizing the stochastic gradient descent, Adam uses a learning rate η for each weight. It additionally calculates the average of the mean and variance of the gradient to find a local minimum fast with low resource consumption (Kingma and Ba 2014). It is important to notice that none of these optimisation algorithms can guarantee finding the global minimum of a function, only the local minimum reachable for the given starting points. Different initial weights can therefore lead to a varying model performance and will be considered during optimisation (Narkhede, Bartakke, and Sutaone 2022).

Besides learning, optimal neural network performance can be achieved by selecting the best hyperparameters of the network. Hyperparameters contain the general information on the neural network, for example, the number of layers, the number of neurons per layer or the batch size (L. Yang and Shami 2020).

Finding the optimal hyperparameters is not a trivial task (L. Yang and Shami 2020). Besides grid search, where every parameter combination is tested, the Bayesian algorithm finds the best hyperparameters by evaluating the choice of parameters after each iteration. First described in Močkus 1975 this algorithm leads to fewer iterations needed in comparison with the grid search or random search algorithm (L. Yang and Shami 2020). Extensive training and hyperparameter optimisation can lead to an overfitting situation. Overfitting occurs when the neural network produces good results on the training set but generalizes worse on new data. The network has not learned the general patterns in the data but rather the explicit data sequence (Bejani and Ghatee 2021).

One simple countermeasure to overfitting is the usage of large training Datasets. If the model trains different situations, the chances of encountering the same ones are reduced (Nielsen 2015). Besides model evaluation, data splitting can be another viable option which is used in this thesis. The whole dataset is split into three different subsets: training data, validation data and test data with 70%, 20% and 10% of the original data accordingly. The training data are used for the normal training procedure and the test data is used to test the performance of the network. The validation data on the other hand is used to validate how the model performs on new data. The loss of the validation data can be calculated after each training epoch. If the validation loss stops decreasing over a period of epochs, the training can be stopped, which is called early stopping (Bejani and Ghatee 2021).

Another option to decrease overfitting is the usage of so-called regularisation terms. In the loss function, a term called weight decay or commonly referred to as L2-regularisation is added to incentivise the usage of smaller weights (Bejani and Ghatee 2021). In the case of the MSE, this gives (Nielsen 2015):

$$\mathrm{MSE}_{\mathrm{L2}} = \mathrm{MSE} + \frac{\lambda}{2n}\sum_{w} w^2 \tag{12}$$

Where λ is the weight decay rate, which can be tuned as an additional hyperparameter.

2.3 LSTM models

Neural networks can solve a variety of complex problems reaching from image recognition (Wu, J. 2017) to natural language processing (Chowdhary 2020) so different network structures have evolved for different tasks.

For forecasting meteorological time series data, Long Short Term Memory Networks (LSTM) have shown as a viable option (Haque, Tabassum, and Hossain 2021, Tran et al. 2021). First described by Hochreiter and Schmidhuber 1997 LSTM networks are a type of recurrent network. The network structure mentioned in the previous chapter called a multi-layer perceptron, can not model temporal sequences, because it lacks the concept of memory (Korstanje 2021).

In recurrent neural networks, RNN, neurons can use previous outputs of itself for the computation of the new output. The recurrent part, which can be conceived as a loop, may also be unfolded. Each recurrent neuron can be imagined as a chain connection, which begins with the first element in the input sequence and ends with the last. The recurrent neuron shares the weights and bias along this unfolded series (Graves 2012). For a longer input series, this can lead to a problem called vanishing/exploding gradient, because each element in the chain gets multiplied by the corresponding weight. If this weight is larger than one the output value of the neuron gets exponentially larger, if it is smaller than one vice versa, which leads to either no training at all or an unstable result (Hochreiter and Schmidhuber 1997).

Long Short Term Memory Networks try to solve this problem by splitting the memory into short and long-term components on the one hand and evaluating which memory to preserve on the other hand. The LSTM cell is therefore split into three gates: The input gate, the output gate and the forget gate (Graves 2012).

The cell gets three different inputs: The previous long-term memory $c_{\text{t-1}}$, the previous short-term memory $a_{\text{t-1}}$ and the input value X_t. Whereas both $c_{\text{t-1}}$ and $a_{\text{t-1}} \in \mathbb{R}^h$ with h being the hidden-size of the network. The complicated cell structure can be again broken down into submodules.

In the first submodule, the forget gate, the long-term memory input is scaled by the input X_t and the short-term memory $a_{\text{t-1}}$ (Graves 2012). This scaling is done by:

$$f_g = \sigma_g\left(w_1 a_{\text{t-1}} + w_2 X_t + b_f\right) \tag{13}$$

and

$$c_{t_s} = f_g \odot c_{\text{t-1}} \tag{14}$$

Where

- σ_g : sigmoid activation function
- w_1, w_2 : forget gate weights
- b_f : forget gate bias
- c_{t_s} : scaled long-term memory
- $\odot$: Hadamard product

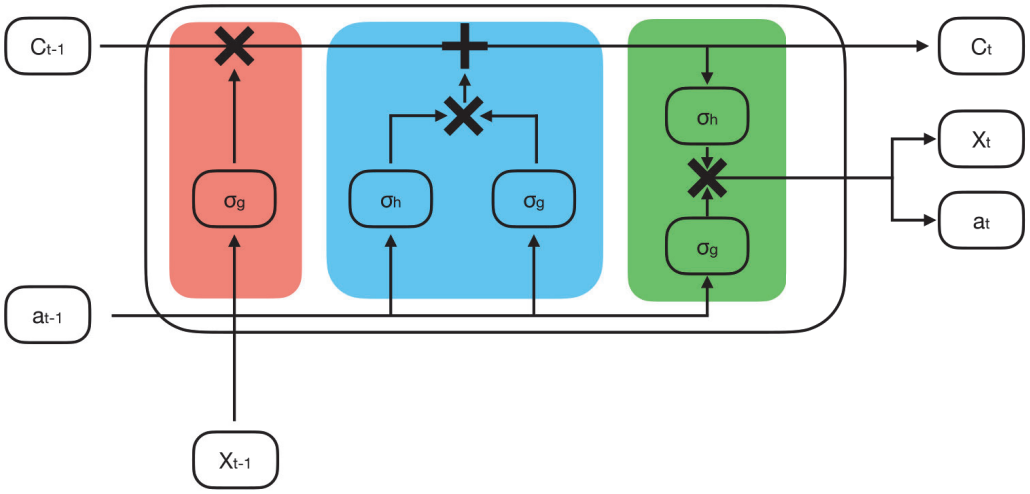


Figure 4: The internal structure of an LSTM cell inspired by Korstanje 2021. Different colours visualize the three gates. In red is the forget gate, in blue the input gate and in green the output gate. The cross symbolizes the Hadamard product and the plus matrix addition. It is worth noting that $a_{\text{t-1}}$ and X_t are concatenated. If arrows are orthogonal on other lines, a copy is produced and not an XOR.

In Equation 14, the Hadamard product, denoted by $\odot$, is a matrix operation (Million 2007). Each element of the matrix A gets multiplied by the corresponding element of matrix B (Million 2007):

$$\begin{pmatrix} A_{1_1} & A_{1_2} & \dots & A_{1_n} \\ A_{2_1} & A_{2_2} & \dots & A_{2_n} \\ \vdots & \vdots & \ddots & \vdots \\ A_{m_1} & A_{m_2} & \dots & A_{m_n} \end{pmatrix} \odot \begin{pmatrix} B_{1_1} & B_{1_2} & \dots & B_{1_n} \\ B_{2_1} & B_{2_2} & \dots & B_{2_n} \\ \vdots & \vdots & \ddots & \vdots \\ B_{m_1} & B_{m_2} & \dots & B_{m_n} \end{pmatrix} = \begin{pmatrix} A_{1_1} \cdot B_{1_1} & A_{1_2} \cdot B_{1_2} & \dots & A_{1_n} \cdot B_{1_n} \\ A_{2_1} \cdot B_{2_1} & A_{2_2} \cdot B_{2_2} & \dots & A_{2_n} \cdot B_{2_n} \\ \vdots & \vdots & \ddots & \vdots \\ A_{m_1} \cdot B_{m_1} & A_{m_2} \cdot B_{m_2} & \dots & A_{m_n} \cdot B_{m_n} \end{pmatrix}$$

In the second submodule, the input gate, the scaled long-term memory from the previous module gets updated by the potential new long-term memory. The potential new long-term memory gets created in a two-step procedure (Graves 2012). First, the new long-term memory needs to be computed by

$$c_{t_{\text{pot}}} = \sigma_g\left(w_3 a_{\text{t-1}} + w_4 X_t + b_{c_{t_{\text{pot}}}}\right) \tag{15}$$

Where

- w_3, w_4 : input gate weights
- $b_{c_{t_{\text{pot}}}}$: potential input gate bias

Then in the second step, the percentage of the new long-term memory that should be added to the old long-term memory gets calculated by

$$c_{t_{\text{per}}} = \sigma_h\left(w_5 a_{\text{t-1}} + w_6 X_t + b_{c_{t_{\text{per}}}}\right) \tag{16}$$

Where

- σ_h : hyperbolic tangent activation function
- w_5, w_6 : input gate weights
- $b_{c_{t_{\text{per}}}}$: precentral input gate bias

Finally, multiplying $c_{t_{\text{pot}}}$ and $c_{t_{\text{per}}}$ and adding them to the scaled long-term memory results in:

$$c_t = c_{t_s} + c_{t_{\text{per}}} \odot c_{t_{\text{pot}}} \tag{17}$$

In the last submodule, the output gate, the new short-term memory of the cell gets computed by using the new long-term memory from the previous submodule. This again is a two-step process analogous to the input gate (Graves 2012).

First, the potential new short-term memory gets computed

$$a_{t_{\text{pot}}} = \sigma_g\left(w_7 c_t + b_{a_{t_{\text{pot}}}}\right) \tag{18}$$

and then the percentage of which to remember gets calculated:

$$a_{t_{\text{per}}} = \tanh_g\left(w_8 a_{\text{t-1}} + w_9 X_t + b_{a_{t_{\text{per}}}}\right) \tag{19}$$

Which gives the new short-term memory

$$a_t = a_{\text{t-1}} + a_{t_{\text{per}}} \odot a_{t_{\text{pot}}} \tag{20}$$

The output X_t of the cell is equal to the new short-term memory a_t (Graves 2012).

3 Methods

This chapter introduces the data and methodology that was used in this thesis to construct and train the forecasting model.

3.1 Input data and preprocessing

To implement and validate the model, data from the Institut für Meteorologie und Klimatologie (IMuK) measuring site Hannover-Herrenhausen in Lower Saxony, Germany, was used. Although measuring a variety of meteorological parameters in high temporal resolution, the measuring site does not comply with the WMO Standards (World Meteorological Organisation 2021). Nearby buildings and greenhouses strongly influence the measurements, but it is nonetheless a good representation of urban climates. The measurements were split between two sites. One is the 50 m tall tower and adjacent measuring field, the other is the roof platform above the institute.

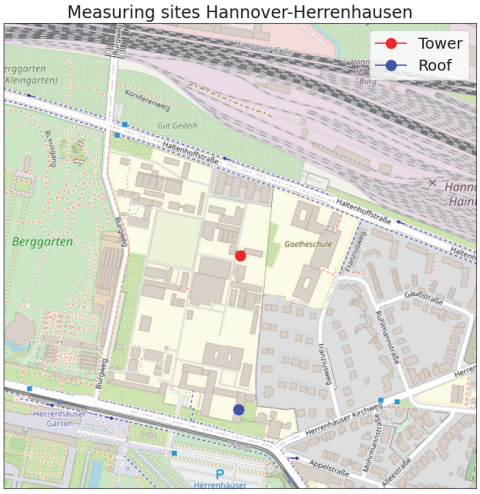


Figure 5: Measuring sites in Hannover-Herrenhausen at 52.39° N and 9.70° E and an elevation of 55 m. Map data from OpenStreetMap was used to create this image. The scale is 1:8000.

Data from the years 2016 to 2022 were used. Table 2 lists all measured parameters, where they were measured and at which height above ground.

Due to measurement device outages, some values were linear interpolated to create a coherent time series. If more than two hours of consecutive measurements were missing of any variable, the whole day was discarded from the training procedure.

Each variable was also resampled from one-minute measurements to one-hour intervals. Using hourly data to forecast meteorological parameters has proven as a good compromise between forecast accuracy and computational resources for forecasting time horizons in the short-term domain (Kreuzer, Munz, and Schlüter 2020, Haque, Tabassum, and Hossain 2021). Resampling was done by calculating the mean values in the resampling interval for all parameters except precipitation, which used the sum of the values.

parameter	location	unit	height [m]
air temperature	Tower	° C	2
relative humidity	Tower	%	2
air pressure	Tower	hPa	2
wind speed	Tower	m/s	10, 50
wind speed peaks	Tower	m/s	10, 50
wind direction	Tower	degree	50
precipitation	Tower	hits	2
global radiation	Roof	W/m^2	10
diffuse radiation	Roof	W/m^2	10

Table 2: Measured values in Hannover-Herrenhausen. All measurements have a temporal resolution of one minute.

Furthermore, some variables were adjusted for the training procedure. The air temperature was recalculated in Kelvin, air pressure in Pa and precipitation in mm. Additionally, air pressure was reduced to sea level height using the following version of the barometric formula (Lente and Katalin 2020):

$$P_0 = P_h \left(1 - \frac{\kappa - 1}{\kappa} \cdot \frac{M \cdot g \cdot (-h)}{R_0 \cdot T_0}\right)^{\frac{\kappa}{\kappa - 1}} \tag{21}$$

Where
- P_h : measured pressure [hPa]
- T_0 : reference temperature [K]
- h : elevation [m]
- R_0 : universal gas constant [$J/(mol \cdot K)$]
- g : gravitational acceleration [m/s^2]
- M : molar mass of dry air [kg/mol]
- κ : ratio of specific heat capacities $\frac{c_p}{c_p}$

For proper training, it is also necessary to normalize the input values and thus scale inputs between 0 and 1 (Sola and Sevilla 1997). This was achieved with the MinMax Scaler by Scikit-learn (Pedregosa et al. 2011). For a time series X, scaling is done like the following (Pedregosa et al. 2011):

$$X_{\text{scaled}} = \frac{X - \min(X)}{\max(X) - \min(X)} \tag{22}$$

And afterwards unscaled in the following way:

$$X_{\text{unscaled}} = X_{\text{scaled}} \cdot (\max(X) - \min(X)) + \min(X) \tag{23}$$

This scaling technique is not viable for spherical units like wind direction. So before normalizing, the wind direction φ was split in u and v components. Along the unit circle, spherical units may be decomposed into their sine and cosine parts (Stover 2023):

$$\varphi_u = \sin\left(\left(\frac{\pi}{180}\right) \cdot \varphi\right) \tag{24}$$

$$\varphi_v = \cos\left(\left(\frac{\pi}{180}\right) \cdot \varphi\right) \tag{25}$$

Figure 17 shows this graphically. This technique was chosen over other methods, e.g. one-hot-encoder, because of implementation simplicity and ease of use.

For the multivariate LSTM, additional derived input parameters were calculated, as shown in Table 3 . These were added to enable smaller window sizes and to explicitly use important synoptical parameters.

parameter	**unit**	**parameter**	**unit**
dew point	K	3h dew point change	K
3h pressure change	Pa	3h precipitation sum	mm
3h temperature sum	K	vertical wind difference	m/s
3h precipitation event	boolean		

Table 3: Additional derived input parameters.

The dew point was approximated using the following formula (Kraus 2007, Anyadike 1984):

$$\begin{aligned} E_{\text{sat}} &= 6.1078 \cdot 10^{\frac{a*T}{T+b}} \\ E &= r \cdot E_{\text{sat}} \\ v &= \log\left(\frac{E}{6.1078}\right) \\ \text{td} &= b \cdot \frac{v}{a-v} \end{aligned} \tag{26}$$

For

$$a \coloneqq \begin{cases} 7.5 \text{ if } T \geq 0 \\ 7.6 \text{ if } T < 0 \text{ over water} \\ 9.5 \text{ if } T < 0 \text{ over ice} \end{cases} \tag{27}$$

and

$$b \coloneqq \begin{cases} 237.3 \text{ if } T \geq 0 \\ 240.7 \text{ if } T < 0 \text{ over water} \\ 265.5 \text{ if } T < 0 \text{ over ice} \end{cases} \tag{28}$$

Where

- E_{sat} : saturation vapour pressure [Pa]
- E : vapour pressure [Pa]
- r : relative humidity
- T : air temperature [K]
- td : calculated dew point temperature

The vertical wind difference was calculated with:

$$\Delta w = w_{50} - w_{10} \tag{29}$$

Where

- w_{50} : windspeed at 50 meters
- w_{10} : windspeed at 10 meters

Finally, the rain event is a boolean value which equals one if there was any precipitation in the last three hours and zero vice versa. Table 12 and Figure 16 give an overview of all available parameters.

The whole dataset from 2016 to 2022 was split into the data for training (2016-2021) and the evaluation/ test data (2022). For the neural networks the data, for training, was then split again into a training data set (70%) and a validation dataset (30%). The splitting was done using Sklearn (Pedregosa et al. 2011).

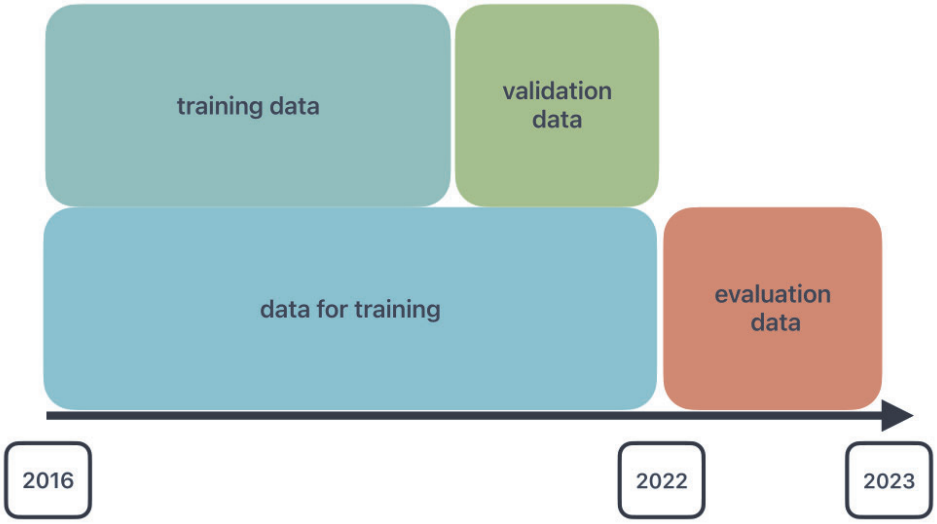


Figure 6: Data splitting for training.

To test the models' ability to generalize to new situations, data from a different location was used in additional tests. The IMuK operates a second measurement field with an adjacent tower in Ruthe, about 30 kilometres south of Hannover, which is shaped by the rural landscape. Although complying with the WMO standards, only a subset of the parameters in Hannover-Herrenhausen is measured in Ruthe, as shown in Table 4.

parameter	unit	height [m]
air temperature	° C	2
relative humidity	%	2
wind speed	m/s	10
wind speed peaks	m/s	10
precipitation	hits	2
global radiation	W/m^2	10

Table 4: Measured values in Ruthe. All measurements have a temporal resolution of one minute.

Similar to the data for Hannover-Herrenhausen, the derived parameters shown in Table 3 were added to the dataset, except for the vertical wind difference and the 3h pressure tendency. The dataset was also prepared in the same way, e.g. resampled and normalized.

The tower stopped operating in mid-2021, so instead of 2022 the year 2020 was used to test the models.

3.2 Used models

This chapter provides a brief overview of the chosen model architectures and training procedures. Generally, five different models were used: one seasonal naive model, one SARIMA model and three models based on LSTM networks. For each of the forecasting variables mentioned in Table 1, there exists one network to forecast this variable. There are therefore 12 submodels per model, totalling in 60 overall submodels.

3.2.1 Baseline approach

The seasonal naive model was used as a baseline approach for model performance evaluation as described in Wilks 2019:

$$Y_{t_{\text{seasonal\_naive}}} = Y_{\text{t-s}} \tag{30}$$

Where s is the seasonal length, so for $s = 24$ hours, the seasonal naive forecast would just use the values from 24 hours ago to make a prediction.

3.2.2 Traditional stochastic method

The SARIMA model was used as a traditional stochastic approach, because of its widespread use and ease of implementation. To find the optimal SARIMA parameters SARIMA $(p, d, q) \times (P, D, Q)_s$ for each forecasting variable, the AUTOARIMA function from the pmdarima package was used which was created by Smith and others 2017.

The AUTOARIMA function finds the combination of parameters, with the minimum Akaike Information Criterion (AIC), first described in Akaike 1974. This criterion gives the relative model performance in relation to the other tested models by evaluating the maximum likelihood (Akaike 1974):

$$\text{AIC} = 2k - 2\ln\left(\hat{L}\right) \tag{31}$$

Where

- k : Number of model parameters
- $\hat{L}$: maximum of the likelihood function

With the usage of the number of model parameters, the AIC tries to prevent the model from overfitting (Akaike 1974). Because of the underlying model restrictions, the seasonal parameter s must be specified for the AUTOARIMA function. A common way to find the seasonality is the Autocorrelation Function (ACF) (Wilks 2019).

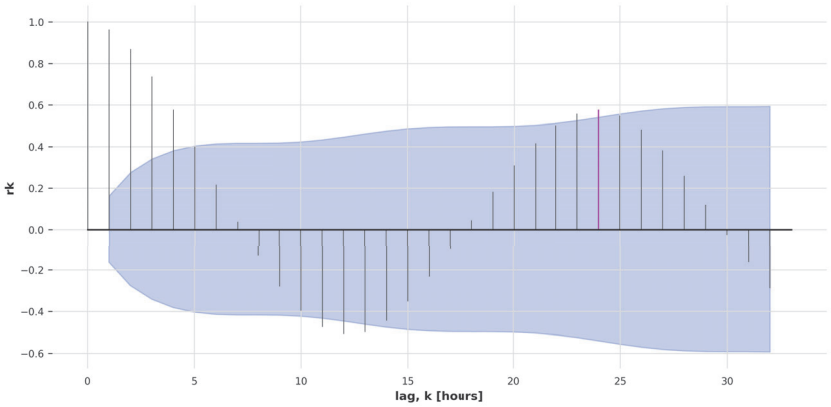


Figure 7: ACF of temperature for different time lags k with the blue field being the $\alpha = 0.05$ confidence interval. The magenta-coloured line shows the most correlation with values that are 24 hours apart, which is to be expected for the temperature.

To find the optimal parameters, the following test was conducted for each forecast variable. The hourly data, of the data for training, was split into windows with a length of 672 hours or 28 days. A window-size of 672 hours was used to capture possible synoptic phenomena and ensure a required long time series for the SARIMA model. For each day in the year, the preceding 672 hours were used as an input time series to calculate the best model parameters with the AUTOARIMA function for a 24-hour forecast. The final parameters for each forecast variable were the median of the yearly parameter distribution from that test and are shown in Table 5.

parameter	non Seasonal parameters	seasonal parameters
Air temperature	$(0,1,1)$	$(0,1,1)_{24}$
Relative humidity	$(0,1,1)$	$(0,1,1)_{24}$
Reduced air pressure	$(0,1,1)$	$(0,1,1)_{24}$
Wind speed_10	$(1,1,1)$	$(0,0,0)_{0}$
Gust speed_10	$(1,1,1)$	$(0,0,0)_{0}$
Wind speed_50	$(0,1,1)$	$(0,1,1)_{24}$
Gust speed_50	$(0,1,1)$	$(0,1,1)_{24}$
Wind direction_sin	$(1,1,1)$	$(0,0,0)_{0}$
Wind direction_cos	$(1,1,1)$	$(0,0,0)_{0}$
Global radiation	$(0,1,1)$	$(0,1,1)_{24}$
Diffuse radiation	$(1,1,1)$	$(0,0,0)_{0}$
Precipitation	$(1,1,1)$	$(0,0,0)_{0}$

Table 5: Non-seasonal parameters (p,d,q) and seasonal parameters $(P,D,Q)_s$ found by the AUTOARIMA function with a sliding window of 672 hours. The seasonality parameter s is displayed in hours.

It is worth noting, that the most common parameter combination in this dataset SARIMA $(0,1,1) \times (0,1,1)_{24}$ is a very common combination in time series forecasting and has earned the title "airline model" based on the forecasting of airline passengers (Box et al. 2015).

The final forecast for each variable for the evaluation data was then again done by a sliding window approach with a window size of 672 values. So in each step, 24 hours were forecasted from the preceding 672 hours. After each prediction, the sliding window was shifted by 24 hours with zero overlap. To prevent unstable predictions, none of the predicted values were used in other predictions. The sliding window slit only over the data for 2022. For ease of use, the Darts package by Herzen et al. 2022 was used to make predictions.

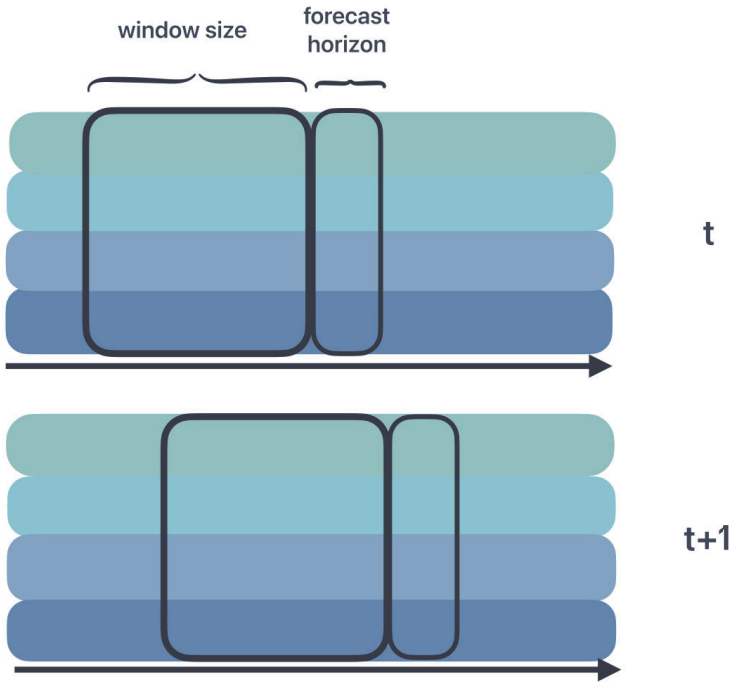


Figure 8: Sliding window approach sliding over the multi-parameter dataset.

3.2.3 LSTM models

Three different LSTM networks were used. A univariate LSTM and two multivariate LSTM. The univariate LSTM, as the SARIMA model, only receives past values of the same variable it forecasts. Contrary, the first multivariate LSTM receives all 20 input parameters (measured + derived as shown in Table 12) whereas the second multivariate LSTM only receives correlating variables. Which variables correlated to the forecasting variable are shown in Figure 9.

If variables show an absolute value of the Pearson correlation coefficient above 0.2 and the p-value is below 0.05, they were used in the prediction of the correlating variable in the correlating multivariate LSTM.

All networks were programmed using the low-level API PyTorch introduced in Paszke et al. 2019 and the mid-level API PyTorch lightning to reduce code boilerplate. The implemented LSTM structure is based on Sak, Senior, and Beaufays 2014.

The univariate LSTM possesses only one input neuron for the one data feature it uses for forecasting. The network consists of a number ($\text{num}_{\text{Layers}}$) of LSTM components stacked on top of each other, where each LSTM component receives the output of the LSTM before as an input. The long-term memory of each LSTM component is a vector with the dimension hidden_size. An example of this structure can be seen in Figure 18. Connected to the last LSTM in the stack is a linear layer. This diffuse layer has 24 output neurons, one for each forecasted hour. The $\text{num}_{\text{Layers}}$ and hidden_size are hyperparameters of this model.

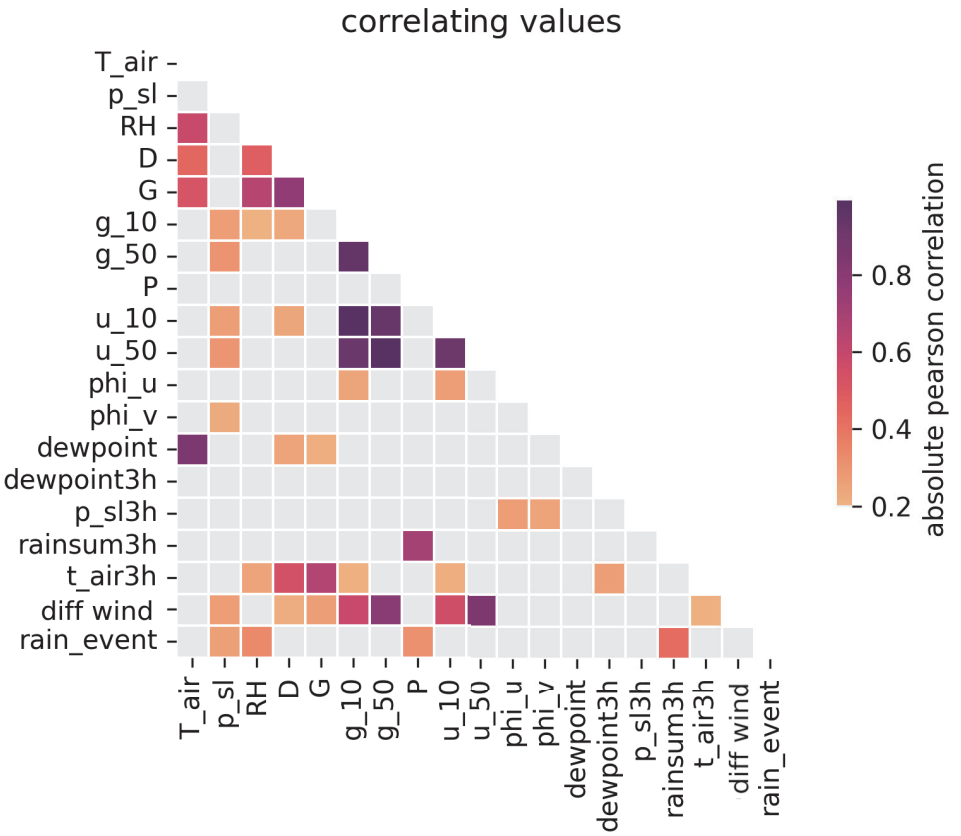


Figure 9: The heatmap shows the absolute correlation matrix for all 20 parameters. The Pearson correlation coefficient was used to calculate the correlation. Darker colours represent stronger correlations. Correlations below 0.2 were coloured light grey.

The multivariate models are structured in the same general fashion, the only difference is the number of input neurons. Whereas the full multivariate model uses 20 input neurons, the number of input neurons for the correlating multivariate model depends on the number of inputs used. Both multivariate LSTM models are so-called Multi Input Single Output (MISO) models. So there is one model for each of the twelve forecasting variables. Although Multi Input Multi Output models (MIMO), exists MISO models achieve better results in meteorological settings (Hewage et al. 2020).

The final forecast was then done again with the sliding window approach over the evaluation data.

3.2.4 Hyperparameter optimisation

The tuning of model hyperparameters is very inefficient if done by hand, as each combination of possible hyperparameters would need to be trained and evaluated afterwards. For large numbers of hyperparameters, this can instead be done with the Optuna package described by Akiba et al. 2019 . Optuna uses the Bayesian algorithm to elevate the model performance after each hyperparameter change and finds those parameters which lead to the smallest loss of the validation data. As evaluation metric was again the MSE used.

Not all hyperparameters were tuned. Table 6 shows all hyperparameters, which were tuned for the models, Table 13 those that were not. The limits of the search space are based on Tran et al. 2021 as well as Kreuzer, Munz, and Schlüter 2020. Also, since these models were trained on a GPU with Cuda acceleration, the available VRAM limits the usage of more complicated models. To try out a variety of parameter combinations, the maximum number of training epochs was also reduced to 20. Additionally, unpromising trials were pruned based on their validation loss.

parameter	possible values
learning rate η	$1 \cdot 10^{-5} < \eta < 1 \cdot 10^{-3}$
weight decay rate λ	$1 \cdot 10^{-5} < \lambda < 1 \cdot 10^{-3}$
dimensions of c_t	$c_t \in [4, 8, 16, 32, 64, 128]$
number of stacked LSTM cells N_{cells}	$N_{\text{cells}} \in [1, 2, 4, 6]$
batch length N	$N \in \{32 + 8n \mid n \in [0, 15]\}$
weight initializer	none, normal distribution, Glorot init., He init.

Table 6: The hyperparameters used in the optimisation process.

Figure 19 shows that the choice of hyperparameters can have a huge impact on the resulting model performance.

For the temperature in the multivariate LSTM, Table 7 shows the optimal values for each hyperparameter found by Optuna. The other hyperparameters for the other variables can be found in the appendix.

parameter	optimal values
learning rate η	0.000425961
weight decay rate λ	0.0000102286
dimensions of c_t	128
number of stacked LSTM cells N_{cells}	2
batch length N	60
weight initializer	kaiming

Table 7: Optimal hyperparameters found by Optuna for the temperature in the multi. LSTM.

3.2.5 Training

As an optimizer during training, the Adam algorithm by Kingma and Ba 2014 was used to find the optimal weights and bias with the MSE as a loss function. Since Adam is based on the stochastic gradient descent, the training procedure was based on the batch approach. The training data was split into batches of the length N. Each batch holds the input with the dimension (L, H_{in}), with L being the sequence length, e.g. the last 24 hours of data and H_{in} being the number of input variables being used. E.g. 1 for the univariate, and up to 20 for the multivariate LSTMs. Each batch has therefore the shape (N, L, H_{in}) The training was overall limited to 200 epochs, where each epoch holds

$$N_{\text{batches}} = \frac{N_{\text{training-data}}}{N} \tag{32}$$

batches. If the loss of the validation data, calculated after each epoch, does not decrease after a patience of 5 epochs, the training was stopped early to prevent overfitting. To further prevent the model from overfitting, L2-regularisation was used inside Adam.

The initial values of the weights and biases can have a significant impact on training convergence (Narkhede, Bartakke, and Sutaone 2022). Different weight initializer have been used during training: none, normal distribution and uniform distribution. The uniform distribution was implemented as Glorot initialization (Glorot and Bengio 2010) and He initialization (He et al. 2015). To ensure comparable models, PyTorch was set to use a deterministic evaluation with random_seed = 45.

3.3 Error metrics

For a final model performance evaluation, the root mean squared error (RMSE) was calculated. The RMSE is defined as the square root of the previously mentioned mean squared error (Wilks 2019):

$$\mathrm{RMSE} = \sqrt{\frac{1}{m}\sum_{\mathrm{i}=1}^{m}\left(Y_{i_{\mathrm{model}}} - Y_{i_{\mathrm{observed}}}\right)^2} \tag{33}$$

The RMSE is a popular choice in determining model performance (Kreuzer, Munz, and Schlüter 2020, Haque, Tabassum, and Hossain 2021, Liang et al. 2021), because it provides a simple and understandable metric. Smaller values characterize better predictions, with $\mathrm{RMSE} = 0$ being a perfect forecast.

Additionally, the Mean absolute scaled error, MASE, was used as a skill score to directly measure each model's improvement over the simple seasonal naive baseline. A MASE below one signals an improvement over the baseline model and vice versa (Hyndman and Athanasopoulos 2018).

$$\mathrm{MASE} = \frac{\sum_{\mathrm{t}=1}^{T}\left|Y_{t_{\mathrm{model}}} - Y_{t_{\mathrm{measured}}}\right|}{\sum_{\mathrm{t}=1}^{T}\left|Y_{t_{\mathrm{seasonal\ naive}}} - Y_{t_{\mathrm{measured}}}\right|} \tag{34}$$

For the precipitation forecast, additionally, the hit rate H and false alarm ratio F were computed (Wilks 2019):

$$\begin{aligned} H &= \frac{a}{a+c} \\ F &= \frac{b}{b+d} \end{aligned} \tag{35}$$

Where a,b,c,d are the number of events in the contingency table, shown in Figure 10

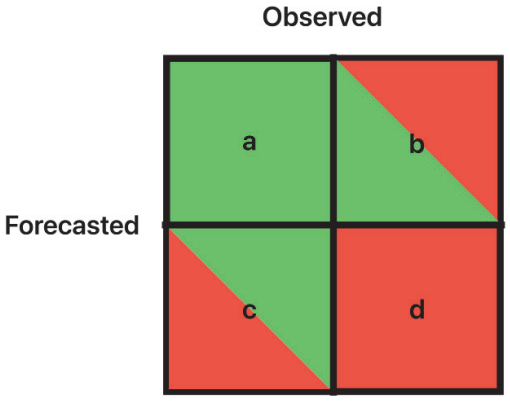


Figure 10: Contingency table inspired by Wilks 2019. Green halves signify true and red false. For example, a is the number of observed and forecasted events and d the number of not forecasted and not observed events

Both range from zero to one, but have diametrically different implications. If $F = 1$ every forecast was a false alarm, and if $H = 1$ every forecast was a correct forecast. It is therefore detrimental for the model to achieve a high hit rate and a low false alarm ratio at the same time. Often H and F are shown together in a so-called H-F-diagram (Wilks 2019), e.g. Figure 14.

4 Results and discussion

This chapter provides the results of this thesis. First, the influence of different forecasting horizons durations and sliding time window lengths were studied. Afterwards, the overall model performance was evaluated, for the whole evaluation dataset and each month. The model performance was then also tested on two meteorological situations and finally deployed on the Ruthe data as a robustness test.

4.1 Influence of different temporal parameters

At first, the influence of the forecasting horizon duration and the sliding time window length, the input data length for the model, were tested. For each permutation of window size and forecast horizon shown in Figure 11, the multivariate LSTM with all input parameters was trained and tested on the evaluation data to calculate the RMSE for the air temperature. Generally, the RMSE increases with the forecast horizon as expected. An increased window-size leads to a lower RMSE up to 24 hours. Larger window-sizes only improve the forecast below a forecast horizon of 18 hours, otherwise, the RMSE stagnates or even gets worse. The model therefore seems to operate optimal with a window size of 24 hours for forecast horizons above 15 hours. Even though not operating optimally below a forecast horizon of 18 hours, the increase in error for a window size of 24 hours is small in comparison to the increased computational complexity of a larger window size. 24 hours could therefore be seen as the optimal window size for all forecast horizons and will be used on all neural networks in this chapter.

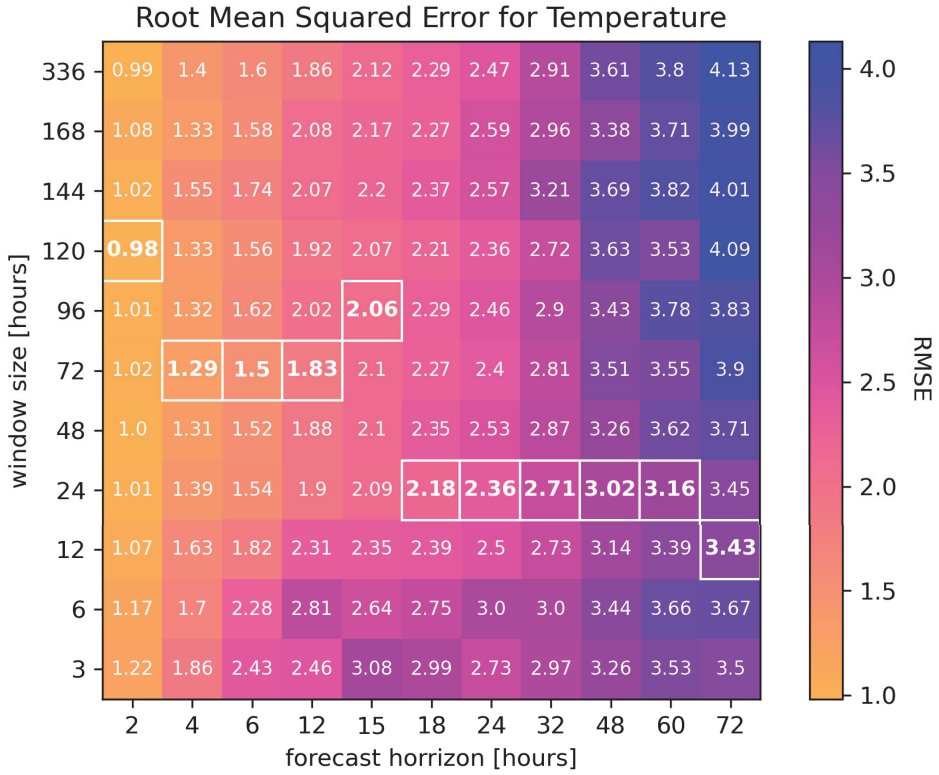


Figure 11: Heatmap showing the RMSE values for different window sizes and forecast horizons. Since 121 different models needed to be trained, the creation of the heatmaps was very resource-heavy and was only done for the temperature. For simplicity, the window size for all LSTM networks was set to 24 hours, although different variables could possess different optima.

4.2 Comparing model performance

To evaluate the model performance, the RMSE and MASE were calculated for the entire evaluation year 2022, Table 8 shows the RMSE values. The three LSTM models combined reach the lowest RMSE values in every parameter category, except precipitation. There is not a one-size-fits-all network solution, rather different networks are required to forecast different parameters. Parameter which meteorological depend on a variety of other parameters, e.g. T_{air}, RH and p_{sl}, show a smaller RMSE for the multivariate LSTM models. On the other hand, parameters that do not depend on other parameters that were observed, e.g. G and D were forecasted best by the univariate LSTM.

Selecting only the correlating parameters leads to some, rather small, model improvements, but this heavily depends on the forecasted parameter. Overall, the multivariate correlating LSTM improved the multivariate LSTM performance only if this model was already performing well and vice versa. Therefore, providing every parameter and letting the LSTM learn on its own which input parameters to use is the proposed tactic.

parameter	seasonal naive	SARIMA	uni. LSTM	multi. LSTM	multi.corr. LSTM
T_{air}	3.277	3.066	2.942	**2.351**	2.516
RH	0.138	0.12	0.116	0.099	**0.098**
p_{sl}	6.588	4.312	3.76	3.539	**3.531**
u_{10}	1.664	1.359	**1.24**	1.277	1.288
u_{50}	2.583	2.01	1.946	**1.812**	1.825
g_{10}	2.388	1.954	1.754	**1.651**	1.678
g_{50}	3.42	2.659	**2.48**	2.513	2.515
φ_u	0.776	0.628	0.543	0.512	**0.51**
φ_v	0.645	0.534	**0.486**	0.528	0.507
P	0.012	**0.009**	0.061	0.062	0.064
G	154.635	127.488	**115.16**	116.418	117.794
D	71.424	93.377	**53.081**	56.648	56.871

Table 8: Comparison of the RMSE values on the evaluation data. Smaller values are an indicator of a better forecast. The bold values characterize the best model for each parameter.

Table 9 shows the MASE values and gives a clearer insight into relative model performance versus the naive seasonal baseline. It generally reproduces the results from Table 8. The LSTM model achieves the lowest MASE values in every parameter, with P again being the only exception. Since both evaluation metrics favour the neural networks, this can be seen as a positive sign. Neither are there larger outliers in the model prediction that would cause higher RMSE values, nor are the errors unevenly distributed, resulting in worse MASE values.

The scale invariance of the MASE allows for a parameter-agnostic model performance analysis. The most improvement over the baseline model was achieved by the multivariate LSTM for p_{sl} with $\text{MASE} = 0.454$. In comparison with the SARIMA model, the largest improvement was achieved in forecasting D with $\Delta\ \text{MASE} = 0.951$. Overall, the neural network performance improvement in comparison with the SARIMA method is satisfactory.

parameter	seasonal naive	SARIMA	uni. LSTM	multi. LSTM	multi.cor. LSTM
T_{air}	1.0	0.881	0.898	**0.694**	0.755
RH	1.0	0.836	0.875	0.733	**0.721**
p_{sl}	1.0	0.562	0.504	**0.454**	0.457
u_{10}	1.0	0.845	**0.747**	0.781	0.794
u_{50}	1.0	0.754	0.744	**0.703**	**0.703**
g_{10}	1.0	0.849	0.74	**0.69**	0.71
g_{50}	1.0	0.751	**0.721**	0.738	0.739
φ_u	1.0	0.823	0.816	**0.724**	0.728
φ_v	1.0	0.835	**0.787**	0.829	0.817
P	**1.0**	1.017	36.583	32.45	38.814
G	1.0	0.897	**0.843**	0.94	0.95
D	1.0	1.811	**0.86**	1.03	1.037

Table 9: Comparison of the MASE values on the evaluation data. Smaller values are an indicator of a better forecast. The bold values characterize the best model for each parameter.

4.2.1 Seasonal influence

To study the influence of seasonality on model performance, the forecasts were split into months and the MASE was calculated for each since most of the forecasted model parameters itself show a yearly seasonality, e.g. G. The usage of RMSE would just mirror the seasonality. Larger values would lead to larger possible squared errors and therefore larger RMSE. MASE does not obliviate these problems but reduces the influence. The study was conducted for the temperature and the results are shown in Figure 12.

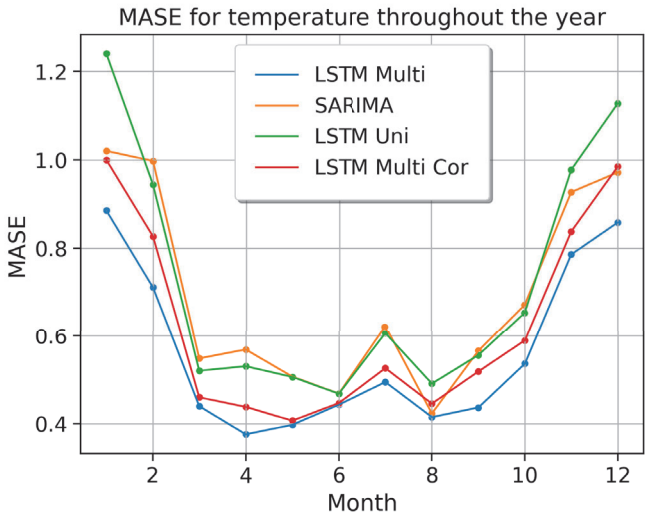


Figure 12: Monthly MASE values for the temperature in 2022.

The MASE changes throughout the year, with lower MASE's in the summer semester and higher values in the winter semester. It is noteworthy, that all models change similar throughout the year and the order of model performance does not change significantly. The full LSTM performs best for each month. The amplitude over the year can be on the one hand attributed to the yearly seasonality and on the other hand to more complex weather situations like inversions that can not be captured by the models. In the summer semester, however, radiation makes temperature easier to forecast.

4.2.2 Heatwave

During the 18th and 23rd of July 2022 a heatwave hit middle Europe and led to very high temperatures. At its peak on the 20th of July, $T_{\text{air}} = 40.4\ °C$ was measured at Hannover-Herrenhausen. This temperature exceeds all temperatures in the data for training and provides a good opportunity to test the models in extreme situations. The model performance was visualized in Figure 13.

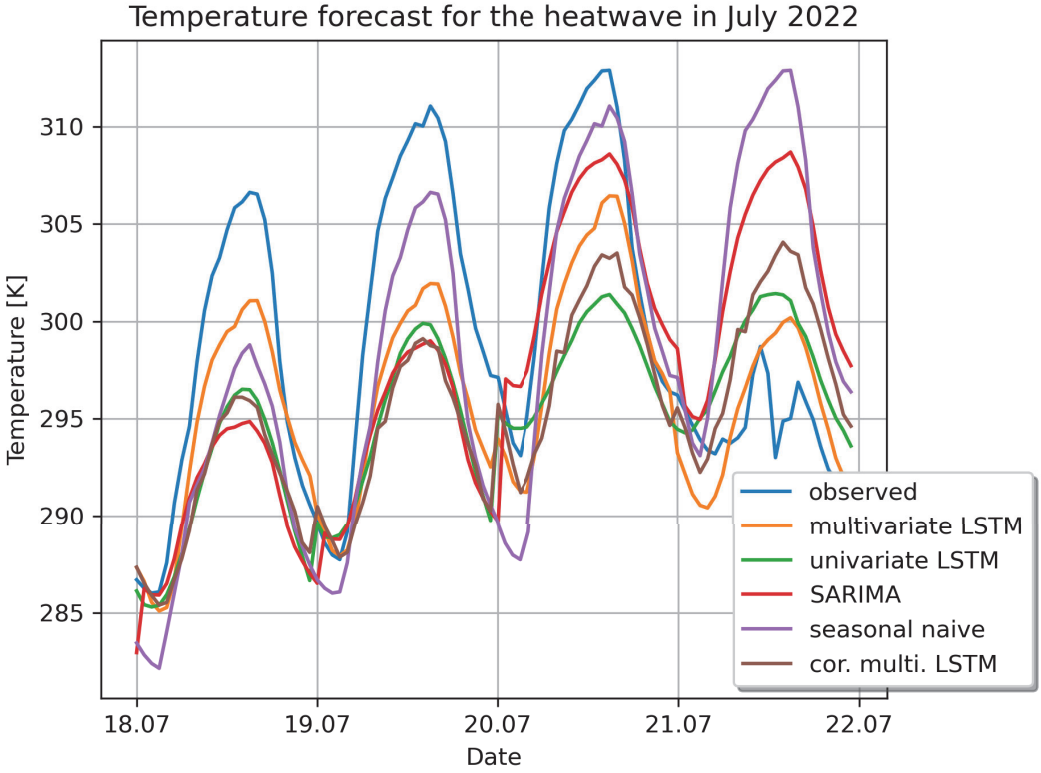


Figure 13: Graphical visualisations of the temperature forecast for all models during the heatwave in July 2022.

None of the models predict the temperature correctly for the 20th, but the seasonal naive gets pretty close, just like the SARIMA model. The good performance of the simple models can be attributed to the steady increase in temperature each day. This can be seen best on the 21st, as the observed temperature abruptly changes, but these models predict nearly the same values as the day before. All three LSTM however correctly predict lower temperatures and a smaller amplitude on this day, with the multivariate LSTM performing best. The model can therefore handle abrupt changes well, but does perform worse on values that are beyond the limits of the training data.

4.2.3 Precipitation

As shown in Table 8 and Table 9 the neural networks as well as the SARIMA model had severe problems in forecasting the precipitation. To inspect this issue further, the H-F diagram was plotted in Figure 14. The univariate LSTM and the multivariate LSTM models had the highest hit rate with $H = 3 \cdot 10^{-2}$ but also the worst false alarm ratio with $F = 0.97$, resulting in only a few correct hits but almost always raising a false alarm. These values can be explained by the second graph in Figure 14, which shows the forecasts for a rain-intense period in December 2022. The training of the networks did not converge. Instead of forecasting the rain events, every model got stuck in an oscillation around a similar fixed value. Since there are only a very few precipitation events, the training data is often, if not most of the time, zero, which can lead to model instability. For this sparse kind of data, LSTM may not be the optimal network architecture. More complex models the combination of convolutional and LSTM

networks, for example, have been more successful in precipitation forecasting (Shi et al. 2015). There could also be a problem with the chosen hyperparameters, as Barrera-Animas et al. 2022 utilize LSTM models more successfully for precipitation forecasts.

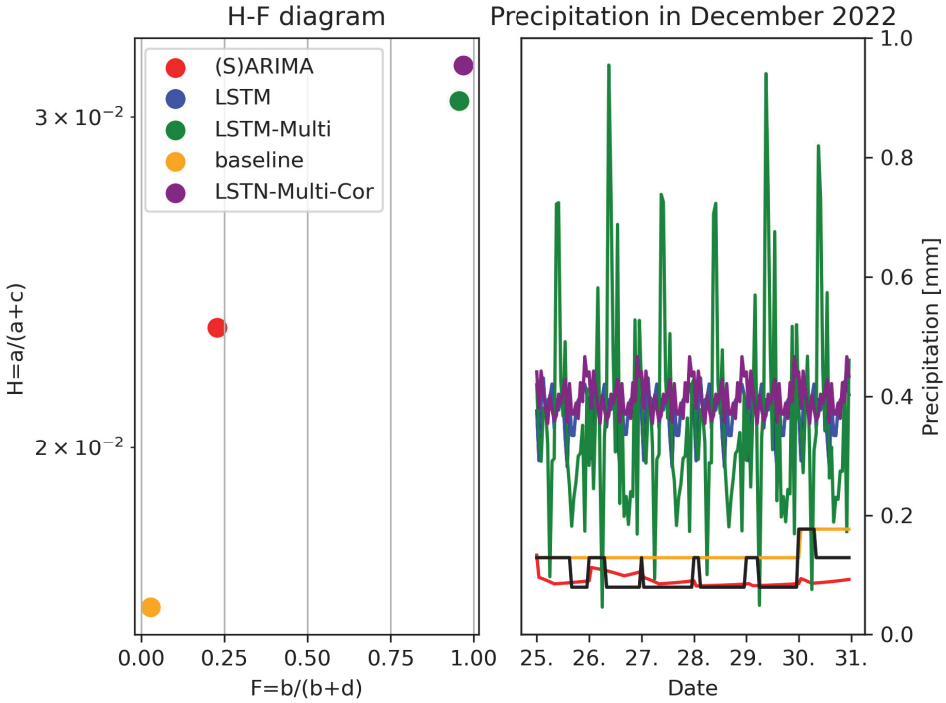


Figure 14: Analysis of precipitation error. The H-F diagram was plotted on the left side and a visual comparison of precipitation forecasts on the right side for December 2022. The observed values are shown in black.

4.3 Testing robustness on Ruthe data

At last, the robustness and therefore the models' ability to generalize should be tested on the Ruthe dataset. Once more the RMSE and MASE values computed for all the available parameters in Ruthe and are shown in Table 10 and Table 11. Ruthe proposed a challenge for every tested model, especially for the T_{air} parameter, where even the naive seasonal model performed significantly worse than in Hannover-Herrenhausen. The multivariate LSTM had to be retrained on the original dataset of Hannover-Herrenhausen with only the parameters described in Section 3.1 as not all original parameters were available in Ruthe, creating a light version of the original model. However, none of the Ruthe data has been used in the training process, as shown in Figure 15.

Instead of the correlating multivariate LSTM, the reduced multivariate LSTM was trained, which just used the observed parameters described in Table 4 and none derived. Overall, the neural networks combined achieved the lowest RMSE values out of all models, with the strengths and weaknesses of each model being similar to the original data. The air temperature marks the only exception. It seems that not every input parameter provided leads to an increase in model performance, as the reduced multivariate LSTM performs significantly better with fewer input parameters than the full multivariate LSTM.

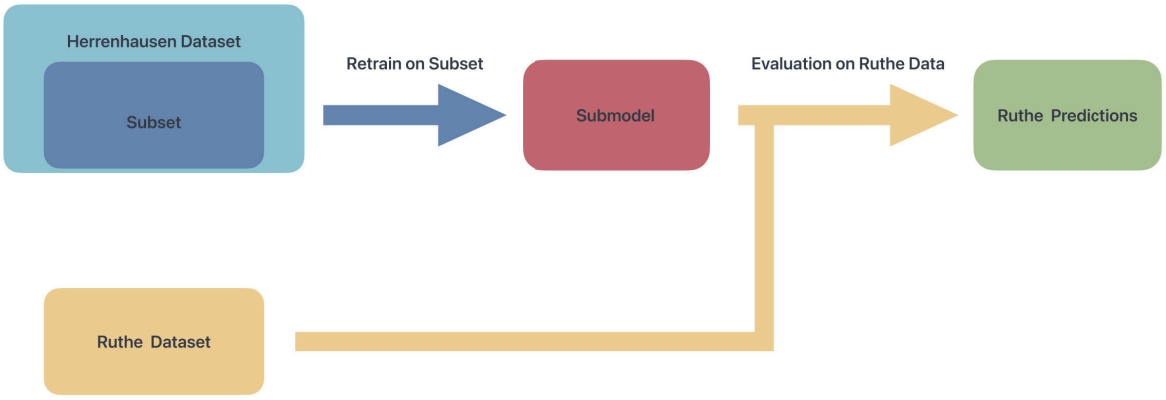


Figure 15: Process for evaluating data on the Ruthe dataset for both multivariate LSTM models. This was necessary because not every original model input parameter was available in Ruthe.

parameter	seasonal naive	SARIMA	uni. LSTM	multi. LSTM	multi.red. LSTM
T_{air}	5.053	5.214	**4.912**	9.614	4.992
RH	0.177	0.17	0.159	0.174	**0.149**
u_{10}	2.637	2.372	1.674	1.604	**1.564**
g_{10}	3.068	2.784	**2.299**	2.406	2.416
P	0.283	0.137	0.108	**0.073**	0.076
G	165.147	140.502	**125.159**	179.133	149.239

Table 10: Comparison of the RMSE values on the evaluation data. Smaller values are an indicator of a better forecast. The bold values characterize the best model for each parameter.

The relative model performance measured by the MASE was worse in almost every parameter, but still provided an acceptable performance increase over the baseline and traditional methods, except for precipitation. The models did therefore generalize pretty well, and the efforts to reduce overfitting were successful.

parameter	seasonal naive	SARIMA	uni. LSTM	multi. LSTM	multi.red. LSTM
T_{air}	1.0	1.03	**0.972**	2.06	0.984
RH	1.0	0.978	0.911	1.007	**0.866**
u_{10}	1.0	0.914	0.664	0.653	**0.626**
g_{10}	1.0	0.917	**0.76**	0.802	0.804
P	**1.0**	1.038	1.731	1.271	1.304
G	1.0	0.892	**0.803**	1.312	1.076

Table 11: Comparison of the MASE values on the evaluation data. Smaller values are an indicator of a better forecast. The bold values characterize the best model for each parameter.

5 Conclusion and future research

The goal of this thesis was to implement and test a forecasting system for all observed parameters at the measuring site Hannover-Herrenhausen. Three different long short term memory networks were composed, one univariate and two multivariate networks for every parameter. The seasonal naive model was used as a baseline model and a SARIMA model to compare against traditional stochastic methods. The hyperparameters were optimized to ensure optimal model performances. RMSE and MASE have been used to evaluate the model performance.

Overall, the neural networks combined showed an improvement in forecasting each parameter except precipitation for both error metrics. The univariate network exceeds in forecasting meteorological parameters that do not depend on other parameters like global radiation, whereas the multivariate network performs best on parameters with multiple meteorological dependencies. The additional correlation test done for the second multivariate LSTM did not always lead to a forecast improvement, sometimes even to a worse performance. This additional step is therefore not advised, instead the LSTM should learn the parameter importance on its own. For the optimal forecast of all parameters, a combination of univariate LSTM and multivariate LSTM should be used. The univariate LSTM to forecast u_{10}, g_{10},φ_u,φ_v, G and D and the multivariate for all other parameters.

Different variations of the forecast horizons and window sizes have been studied to find the optimal combination. For most forecast horizons in the short term domain, a window size of the past 24 hours strikes the optimal balance between model performance and complexity for the air temperature. Future research is needed to validate this for the other parameters. The networks showed a good reaction to abrupt changes in the data, but struggled if the data was beyond the limits of the training data, as shown with the heatwave in Section 4.2.2. In the context of climate change, this presents a significant model limitation and needs further research. Precipitation forecast was another struggle for the neural networks, and other models or different data preprocessing methods should be studied to achieve better performance. The robustness test in Ruthe revealed some flaws in the multivariate temperature model but showed that the models generalize pretty well overall. To test the model's robustness, a light version of the original model was used to comply with the lack of several model parameters. Another interesting approach for further research could be the usage of the full pre-trained Hannover-Herrenhausen model on the Ruthe data, with filler values in the dataset for the empty parameters.

Since only the training of the model requires time and high computational resources, but not the usage to make a forecast, meteorological neural networks, can be deployed on low-end hardware like smartphones or even Raspberry Pi. As shown by Hewage et al. 2020. Although, as smartphone performance increases, they may be used in the future to even train these models locally to make forecasts in remote locations. As the computational resources were limited for this thesis, it would be fascinating to test out more complex models like the Transformer models described in Vaswani et al. 2017. Additionally, the usage of multiple weather stations as input data could provide a fascinating opportunity to develop more robust models.

Advancements in meteorology were often tied to advancements in technology. As we accelerate the progress in machine learning and stand at the precipice of a new era, this thesis tried to shed some light on the underlying processes.

Bibliography

Akaike, H. 1974. *A New Look at the Statistical Model Identification*. IEEE Transactions on Automatic Control. Vol. 19. no. 6. n.p., 716–723, https://doi.org/10.1109/TAC.1974.1100705.

Akiba, Takuya et al. 2019. *Optuna: A Next-Generation Hyperparameter Optimization Framework*. Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.

Alerskans, Emy et al. 2022. *A Transformer Neural Network for Predicting near-Surface Temperature*. Meteorological Applications. Vol. 29. no. 5. Wiley Online Library.

Anyadike, RN C. 1984. *Assessment of Various Formulae for the Computation of Saturation Vapour Pressures over Liquid Water*. Archiv Für Meteorologie, Geophysik Und Bioklimatologie. Serie a, Meteorologie Und Geophysik. Vol. 33. 2-3. Springer, 239–243.

Barrera-Animas, Ari Yair et al. 2022. *Rainfall Prediction: A Comparative Analysis of Modern Machine Learning Algorithms for Time-Series Forecasting*. Machine Learning with Applications 7. Elsevier, 100204.

Bejani, Mohammad Mahdi, and Mehdi Ghatee. 2021. *A Systematic Review on Overfitting Control in Shallow and Deep Neural Networks*. Artificial Intelligence Review. Springer, 1–48.

Bottou, Léon. 1998. *Online Algorithms and Stochastic Approximations*. Online Learning in Neural Networks. Cambridge University Press.

Bottou, Léon. 2012. "Stochastic Gradient Descent Tricks." In *Neural Networks: Tricks of the Trade: Second Edition*. Springer, 421–436.

Box, G., G. Jenkins, and G. Reinsel. 1970. *Time Series Analysis: Forecasting and Control Holden-Day San Francisco*. Boxtime Series Analysis: Forecasting and Control Holden Day1970. n.p.

Box, G. et al. 2015. *Time Series Analysis: Forecasting and Control*. John Wiley & Sons.

Chowdhary, KR. 2020. *Natural Language Processing*. Fundamentals of Artificial Intelligence. Springer, 603–649.

Desolte, Timothy, and Michael K. Tippet. 2022. *Statistical Methods for Climate Scientist*. Cambridge University Press, https://doi.org/10.1017/9781108659055.

Eide, S. et al. 2022. *Deep Tower Networks for Efficient Temperature Forecasting from Multiple Data Sources*. Sensors. Vol. 22. no. 7. MDPI, 2802.

Eide, S. et al. 2021. *Temperature Forecasting Using Tower Networks*. Proceedings of the 2021 Workshop on Intelligent Cross-Data Analysis and Retrieval, 18–23.

Glorot, Xavier, and Yoshua Bengio. 2010. *Understanding the Difficulty of Training Deep Feedforward Neural Networks*. Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, 249–256.

Graves, Alex. 2012. *Supervised Sequence Labelling with Recurrent Neural Networks, Studies in Computational Intelligence*. Berlin: Springer, https://doi.org/10.1007/978-3-642-24797-2.

Haque, Ehtashamul, Sanzana Tabassum, and Eklas Hossain. 2021. *A Comparative Analysis of Deep Neural Networks for Hourly Temperature Forecasting*. IEEE Access 9. IEEE, 160646–160660.

He, Kaiming et al. 2015. *Delving Deep into Rectifiers: Surpassing Human-Level Performance on Imagenet Classification*. Proceedings of the IEEE International Conference on Computer Vision, 1026–1034.

Herzen, Julien et al. 2022. *Darts: User-Friendly Modern Machine Learning for Time Series*. Journal of Machine Learning Research. Vol. 23. no. 124, 1–6, http://jmlr.org/papers/v23/21-1177.html.

Hewage, Pradeep et al. 2020. *Temporal Convolutional Neural (TCN) Network for an Effective Weather Forecasting Using Time-Series Data from the Local Weather Station*. Soft Computing 24. Springer, 16453–16482.

Hochreiter, Sepp, and Jürgen Schmidhuber. 1997. *Long Short-Term Memory*. Neural Computation. Vol. 9. no. 8, 1735–1780, https://doi.org/10.1162/neco.1997.9.8.1735.

Hudspeth, A. J. et al. 2013. *Principles of Neural Science*. New York: McGraw-Hill Medical.

Hyndman, Rob J, and George Athanasopoulos. 2018. *Forecasting: Principles and Practice*. OTexts.

Khashei, Mehdi, Mehdi Bijari, and Seyed Reza Hejazi. 2012. *Combining Seasonal ARIMA Models with Computational Intelligence Techniques for Time Series Forecasting*. Soft Computing 16. Springer, 1091–1105.

Kingma, Diederik P, and Jimmy Ba. 2014. *Adam: A Method for Stochastic Optimization*. Arxiv Preprint Arxiv:1412.6980.

Korstanje, Joos. 2021. *Advanced Forecasting with Python : With State-of-the-Art-Models Including Lstms, Facebook's Prophet, and Amazon's Deepar, Springer Ebook Collection*. New York: Apress, https://doi.org/10.1007/978-1-4842-7150-6.

Kraus, Helmut. 2007. *Die Atmosphäre Der Erde: Eine Einführung in Die Meteorologie*. Springer-Verlag.

Kreuzer, David, Michael Munz, and Stephan Schlüter. 2020. *Short-Term Temperature Forecasts Using a Convolutional Neural Network—an Application to Different Weather Stations in Germany*. Machine Learning with Applications 2. Elsevier, 100007.

Lente, Gábor, and Ösz Katalin. 2020. *Barometric Formulas:various Derivations and Comparisons to Environmentally Relevant Observations*. Chemtexts. Vol. 13. no. 6. Springer, 1–14, https://doi.org/10.1007/s40828-020-0111-6.

Liang, Shuo et al. 2021. *Method of Bidirectional LSTM Modelling for the Atmospheric Temperature*. Intelligent Automation & Soft Computing. Vol. 30. no. 2.

Linnainmaa, Seppo. 1976. *Taylor Expansion of the Accumulated Rounding Error*. BIT Numerical Mathematics. Vol. 16. no. 2. n.p.: Springer, 146–160.

Million, Elizabeth. 2007. *The Hadamard Product*. Course Notes. Vol. 3. no. 6, 1–7.

Močkus, Jonas. 1975. *On Bayesian Methods for Seeking the Extremum*. Optimization Techniques IFIP Technical Conference: Novosibirsk, July 1--7, 1974. n.p., 400–404.

Narkhede, Meenal V, Prashant P Bartakke, and Mukul S Sutaone. 2022. *A Review on Weight Initialization Strategies for Neural Networks*. Artificial Intelligence Review. Vol. 55. no. 1. Springer, 291–322.

Nielsen, Michael A. 2015. *Neural Networks and Deep Learning*. Determination Press.

Paszke, Adam et al. 2019. *Pytorch: An Imperative Style, High-Performance Deep Learning Library*.

Pedregosa, F. et al. 2011. *Scikit-Learn: Machine Learning in Python*. Journal of Machine Learning Research 12, 2825–2830.

Rosenblatt, Frank. 1958. *The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain.* Psychological Review. Vol. 65. no. 6. n.p.: American Psychological Association, 386.

Ruder, Sebastian. 2016. *An Overview of Gradient Descent Optimization Algorithms.* Arxiv Preprint Arxiv: 1609.04747.

Rumelhart, David E, Geoffrey E Hinton, and Ronald J Williams. 1986. *Learning Representations by Back-Propagating Errors.* Nature. Vol. 323. no. 6088. Nature Publishing Group UK London, 533–536.

Sak, Haşim, Andrew Senior, and Françoise Beaufays. 2014. *Long Short-Term Memory Based Recurrent Neural Network Architectures for Large Vocabulary Speech Recognition.*

Samal, K Krishna Rani et al. 2019. *Time Series Based Air Pollution Forecasting Using SARIMA and Prophet Model.* Proceedings of the 2019 International Conference on Information Technology and Computer Communications, 80–85.

Sharma, S., S. Sharma, and A. Athaiya. 2017. *Activation Functions in Neural Networks.* Towards Data Sci. Vol. 6. no. 12, 310–316.

Shi, X et al. 2015. *Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting.* Advances in Neural Information Processing Systems 28.

Smith, Taylor G., and others. 2017. "Pmdarima: ARIMA Estimators for Python," accessed August 27, 2023, https://www.alkaline-ml.com/pmdarima.

Sola, J., and J. Sevilla. 1997. *Importance of Input Data Normalization for the Application of Neural Networks to Complex Industrial Problems.* IEEE Transactions on Nuclear Science. Vol. 44. no. 3, 1464–1468, https://doi.org/10.1109/23.589532.

Stover, Christopher. 2023. "Unit Circle from Mathworld Alpha Wolfram Web Resource, Created by Eric W. Weisstein.," accessed August 23, 2023, https://mathworld.wolfram.com/UnitCircle.html.

Tena Garcia, Jorge Luis et al. 2019. *Forecast of Daily Output Energy of Wind Turbine Using Sarima and Nonlinear Autoregressive Models.* Advances in Mechanical Engineering. Vol. 11. no. 2. SAGE Publications Sage UK: London, England.

Tran, T. T. K. et al. 2021. *A Review of Neural Networks for Air Temperature Forecasting.* Water. Vol. 13. no. 9. MDPI, 1294.

Vaswani, Ashish et al. 2017. *Attention Is All You Need.* Advances in Neural Information Processing Systems 30.

Von Bartheld, Christopher S, Jami Bahney, and Suzana Herculano-Houzel. 2016. *The Search for True Numbers of Neurons and Glial Cells in the Human Brain: A Review of 150 Years of Cell Counting.* Journal of Comparative Neurology. Vol. 524. no. 18. Wiley Online Library, 3865–3895.

Whittle, Peter. 1951. *Hypothesis Testing in Time Series Analysis.* n.p.

Wilks, Daniel S. 2019. *Statistical Methods in the Atmospheric Sciences.* Elsevier, 485–540, https://doi.org/10.1016/B978-0-12-815823-4.00002-X.

World Meteorological Organisation. 2021. *Guide to Instruments and Methods of Observation Volume I: Measurement of Meteorological Variables.* Ed. Publications Board Chair. 7 bis, avenue de la Paix,P.O. Box 2300,CH-1211 Geneva 2, Switzerland, 584.

Wu, J. 2017. *Introduction to Convolutional Neural Networks.* National Key Lab for Novel Software Technology. Nanjing University. China. Vol. 5. no. 23, 495.

Xu, Shuojiang, Hing Kai Chan, and Tiantian Zhang. 2019. *Forecasting the Demand of the Aviation Industry Using Hybrid Time Series SARIMA-SVR Approach.* Transportation Research Part E: Logistics and Transportation Review 122. Elsevier, 169–180.

Yang, L., and A. Shami. 2020. *On Hyperparameter Optimization of Machine Learning Algorithms: Theory and Practice.* Neurocomputing 415. Elsevier, 295–316.

Appendix

Input data for the multivariate LSTM

parameter	unit	mean	variance
air temperature	° C	284.54	7.79
relative humidity	%	75	17
reduced air pressure	Pa	101570,27	976.06
wind speed_10	m/s	2.08	1.3
gust speed_10	m/s	2.97	1.87
wind speed_50	m/s	4.46	2.01
gust speed_50	m/s	5.80	2.74
wind direction_sin	radian	-0.17	0.74
wind direction_cos	radian	-0.30	0.57
global radiation	W/m^2	118.91	201.77
diffuse radiation	W/m^2	58.76	95.17
precipitation	mm	0.07	0.42
dew point	K	279.66	6.1
3h dew point change	K	0	0.37
3h pressure change	Pa	-0.01	43.09
3h precipitation sum	mm	0	0.03
3h temperature change	K	0	0.83
vertical wind difference	m/s	2.38	0.95
3h precipitation event	boolean	0.00	0.13

Table 12: Overview of the training parameters.

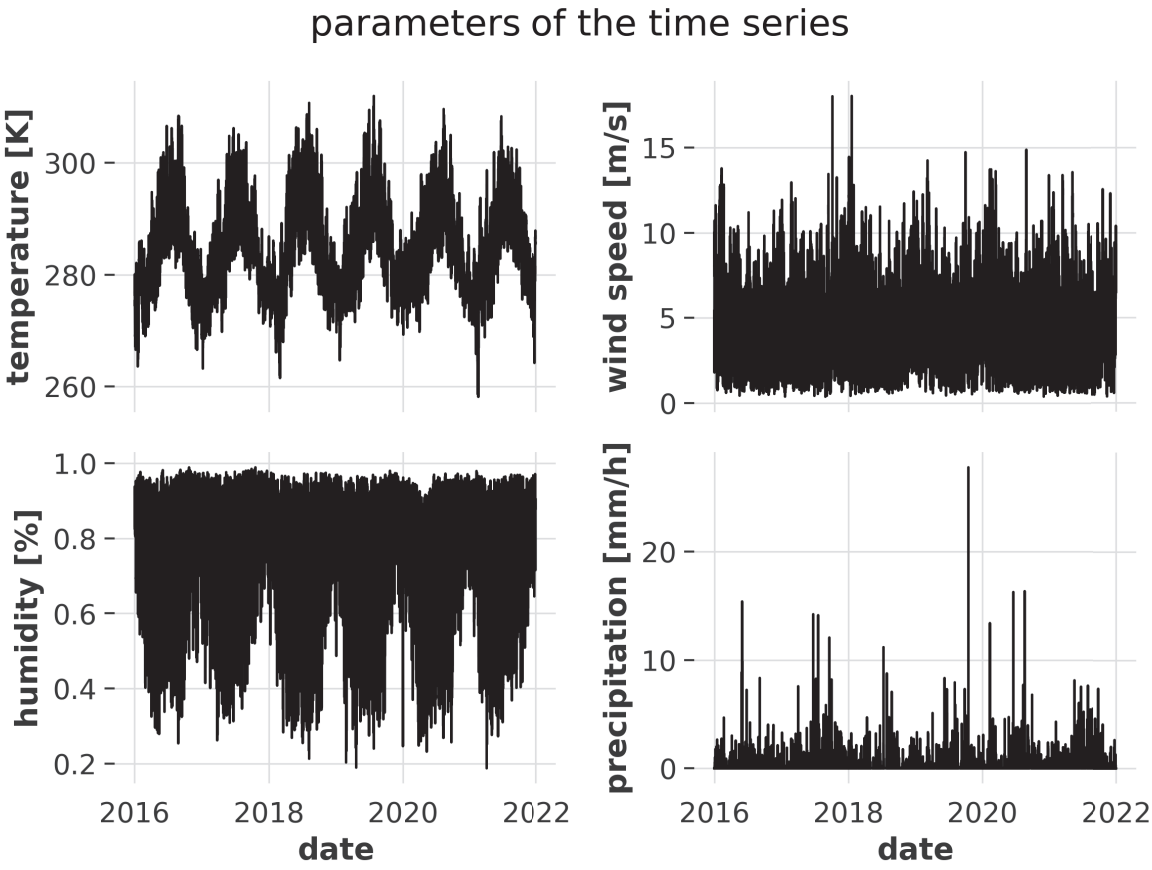


Figure 16: Graphical representation of the data for training. Shown are the temperature, humidity, precipitation, and wind speed for the years 2016-2021.

Splitting wind direction

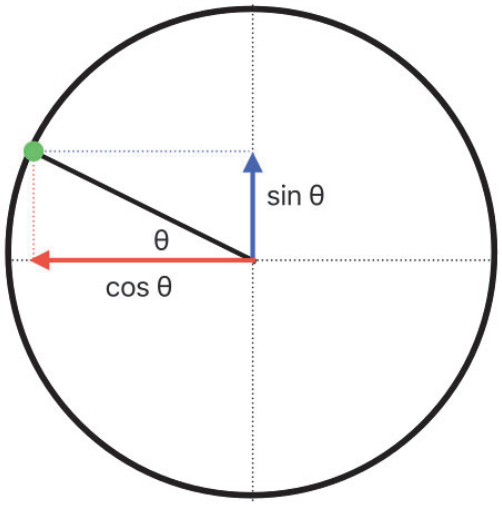


Figure 17: Splitting an angle θ into the sin and cos components along a unit circle, with $r = 1$. Inspired by Stover 2023.

model structure

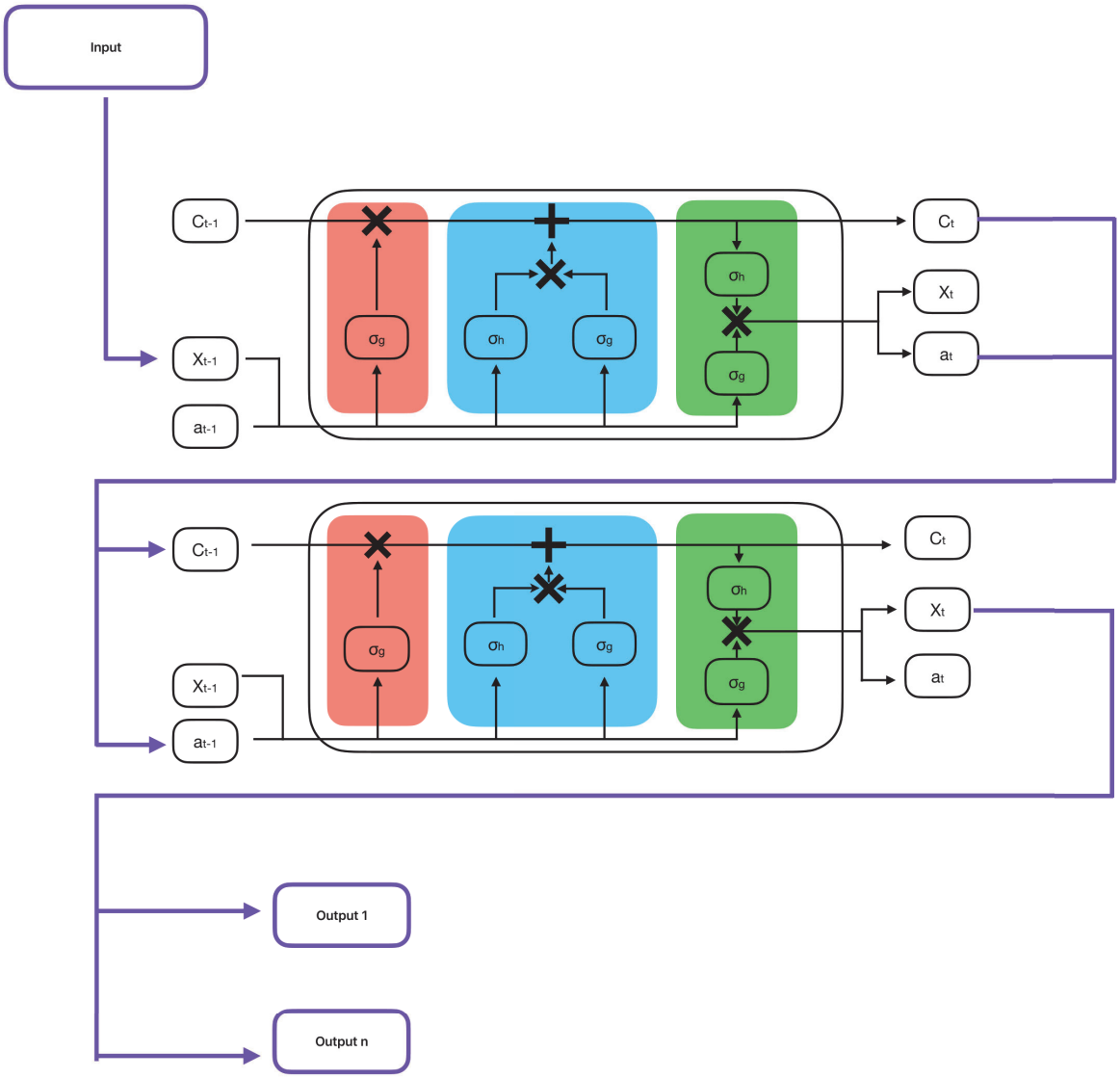


Figure 18: General structure of the stacked LSTM approach with $N_{\text{cells}} = 2$. The input gets used to compute the current long-term memory C_t and the short term memory a_t. Both get then used as additional inputs to the second LSTM cell, which then computes C_t and the short-term memory a_t again. X_t, the output from the second cell, gets used by a linear diffuse layer. For simplicity, only 2 of 24 diffuse output neurons are shown.

Hyperparameters

hyperparameter	univariate LSTM	multi. LSTM	corr. multivariate LSTM
optimizer	Adam	Adam	Adam
inputs	1	20	1-20
dropout	0	0	0
patience	5	5	5
training size	70%	70%	70%
validation size	30%	30%	30%
window shift	24 h	24 h	24 h
window size	24 h	24 h	24 h
epochs	200	200	200

Table 13: Hyperparameters that were not optimized.

Univariate LSTM

humidity

parameter	optimal values
learning rate η	0.0004421658
weight decay rate λ	0.0002617564
dimensions of c_t	32
number of stacked LSTM cells N_{cells}	1
batch length N	60
weight initializer	xavier

Table 14: Optimal hyperparameters found by Optuna for the humidity in the uni. LSTM.

reduced pressure

parameter	optimal values
learning rate η	0.0000244618
weight decay rate λ	0.0000239736
dimensions of c_t	128
number of stacked LSTM cells N_{cells}	2
batch length N	60
weight initializer	kaiming

Table 15: Optimal hyperparameters found by Optuna for the reduced pressure in the uni. LSTM.

global radiation

parameter	optimal values
learning rate η	0.0002392531
weight decay rate λ	0.000034903
dimensions of c_t	64
number of stacked LSTM cells N_{cells}	4
batch length N	60
weight initializer	kaiming

Table 16: Optimal hyperparameters found by Optuna for the global radiation in the uni. LSTM.

diffuse radiation

parameter	optimal values
learning rate η	0.0004733867
weight decay rate λ	0.0000606423
dimensions of c_t	8
number of stacked LSTM cells N_{cells}	2
batch length N	60
weight initializer	kaiming

Table 17: Optimal hyperparameters found by Optuna for the diffuse radiation in the uni. LSTM.

precipitation

parameter	optimal values
learning rate η	0.0001622737
weight decay rate λ	0.0001217253
dimensions of c_t	4
number of stacked LSTM cells N_{cells}	2
batch length N	60
weight initializer	kaiming

Table 18: Optimal hyperparameters found by Optuna for the precipitation in the uni. LSTM.

wind 10 m

parameter	optimal values
learning rate η	0.0007976957
weight decay rate λ	0.0000485815
dimensions of c_t	32
number of stacked LSTM cells N_{cells}	4
batch length N	60
weight initializer	kaiming

Table 19: Optimal hyperparameters found by Optuna for the wind 10 m in the uni. LSTM.

wind 50 m

parameter	optimal values
learning rate η	0.0000720454
weight decay rate λ	0.000130051
dimensions of c_t	4
number of stacked LSTM cells N_{cells}	1
batch length N	60
weight initializer	normal

Table 20: Optimal hyperparameters found by Optuna for the wind 50 m in the uni. LSTM.

gust 10 m

parameter	optimal values
learning rate η	0.000025003
weight decay rate λ	0.000011801
dimensions of c_t	64
number of stacked LSTM cells N_{cells}	2
batch length N	60
weight initializer	kaiming

Table 21: Optimal hyperparameters found by Optuna for the gust 10 m in the uni. LSTM.

gust 50 m

parameter	optimal values
learning rate η	0.0000295239
weight decay rate λ	0.0000130325
dimensions of c_t	128
number of stacked LSTM cells N_{cells}	6
batch length N	60
weight initializer	kaiming

Table 22: Optimal hyperparameters found by Optuna for the gust 50 m in the uni. LSTM.

wind direction sin

parameter	optimal values
learning rate η	0.0009123493
weight decay rate λ	0.0000166529
dimensions of c_t	16
number of stacked LSTM cells N_{cells}	1
batch length N	60
weight initializer	normal

Table 23: Optimal hyperparameters found by Optuna for the wind direction sin in the uni. LSTM.

wind direction cos

parameter	optimal values
learning rate η	0.0000615112
weight decay rate λ	0.0000212533
dimensions of c_t	128
number of stacked LSTM cells N_{cells}	1
batch length N	60
weight initializer	xavier

Table 24: Optimal hyperparameters found by Optuna for the wind direction cos in the uni. LSTM.

Multivariate LSTM

humidity

parameter	optimal values
learning rate η	0.0002968177
weight decay rate λ	0.0000682716
dimensions of c_t	64
number of stacked LSTM cells N_{cells}	4
batch length N	40
weight initializer	kaiming

Table 25: Optimal hyperparameters found by Optuna for the humidity in the multi. LSTM.

reduced pressure

parameter	optimal values
learning rate η	0.0002633076
weight decay rate λ	0.0000273408
dimensions of c_t	128
number of stacked LSTM cells N_{cells}	1
batch length N	32
weight initializer	kaiming

Table 26: Optimal hyperparameters found by Optuna for the reduced pressure in the multi. LSTM.

global radiation

parameter	optimal values
learning rate η	0.0000242157
weight decay rate λ	0.0002955667
dimensions of c_t	128
number of stacked LSTM cells N_{cells}	2
batch length N	40
weight initializer	xavier

Table 27: Optimal hyperparameters found by Optuna for the global radiation in the multi. LSTM.

diffuse radiation

parameter	optimal values
learning rate η	0.0000198443
weight decay rate λ	0.0001357626
dimensions of c_t	128
number of stacked LSTM cells N_{cells}	2
batch length N	48
weight initializer	xavier

Table 28: Optimal hyperparameters found by Optuna for the diffuse radiation in the multi. LSTM.

precipitation

parameter	optimal values
learning rate η	0.001
weight decay rate λ	0.0000102286
dimensions of c_t	1
number of stacked LSTM cells N_{cells}	2
batch length N	32
weight initializer	kaiming

Table 29: Optimal hyperparameters found by Optuna for the precipitation in the multi. LSTM.

wind 10 m

parameter	optimal values
learning rate η	0.0002728777
weight decay rate λ	0.0008340642
dimensions of c_t	128
number of stacked LSTM cells N_{cells}	1
batch length N	56
weight initializer	xavier

Table 30: Optimal hyperparameters found by Optuna for the wind 10 m in the multi. LSTM.

wind 50 m

parameter	optimal values
learning rate η	0.0001669097
weight decay rate λ	0.0000228338
dimensions of c_t	64
number of stacked LSTM cells N_{cells}	1
batch length N	104
weight initializer	xavier

Table 31: Optimal hyperparameters found by Optuna for the wind 50 m in the multi. LSTM.

gust 10 m

parameter	optimal values
learning rate η	0.0005651945
weight decay rate λ	0.0000135455
dimensions of c_t	128
number of stacked LSTM cells N_{cells}	6
batch length N	104
weight initializer	kaiming

Table 32: Optimal hyperparameters found by Optuna for the gust 10 m in the multi. LSTM.

gust 50 m

parameter	optimal values
learning rate η	0.0004698578
weight decay rate λ	0.0001099339
dimensions of c_t	32
number of stacked LSTM cells N_{cells}	4
batch length N	104
weight initializer	kaiming

Table 33: Optimal hyperparameters found by Optuna for the gust 50 m in the multi. LSTM.

wind direction sin

parameter	optimal values
learning rate η	0.0007038974
weight decay rate λ	0.0000258223
dimensions of c_t	64
number of stacked LSTM cells N_{cells}	1
batch length N	32
weight initializer	xavier

Table 34: Optimal hyperparameters found by Optuna for the wind direction sin in the multi. LSTM.

wind direction cos

parameter	optimal values
learning rate η	0.000717075
weight decay rate λ	0.000017122
dimensions of c_t	64
number of stacked LSTM cells N_{cells}	4
batch length N	32
weight initializer	kaiming

Table 35: Optimal hyperparameters found by Optuna for the wind direction cos in the multi. LSTM.

Automatic hyperparametrisiation

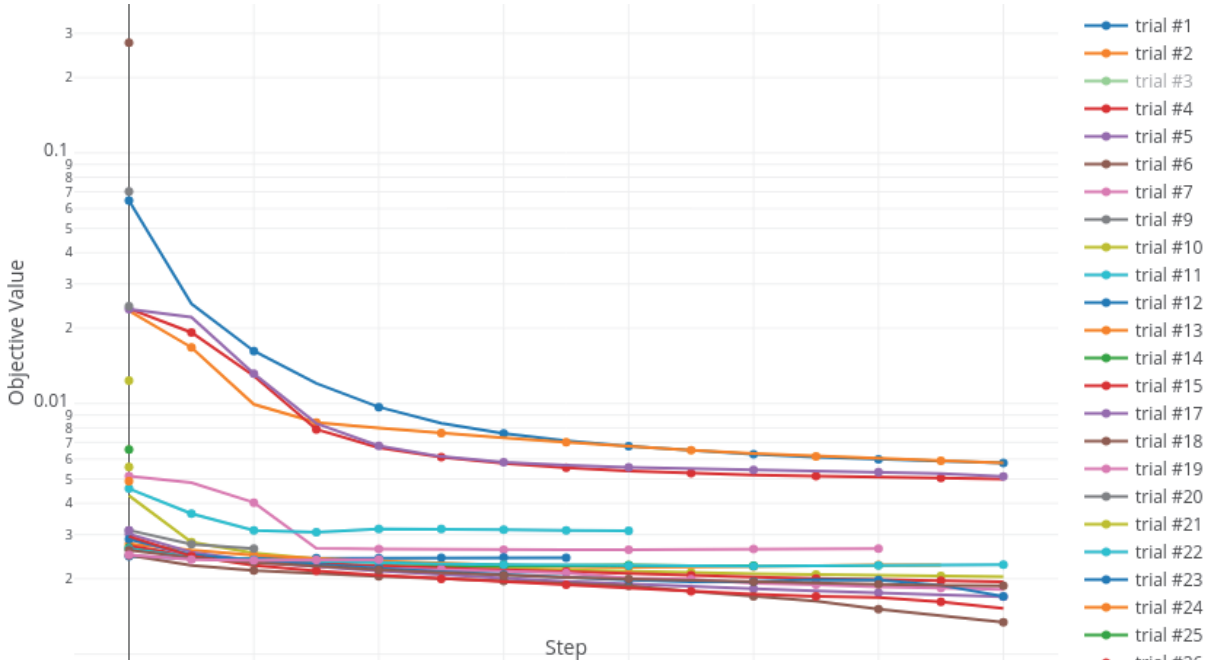


Figure 19: Graphic produced by Optuna. For the multivariate LSTM forecasting temperature, the different trials are shown. On the x-axis are the number of elapsed training epochs and on the y-axis the MSE of the validation data on a logarithmic scale.

Table index

Figure index

Glossary

ACF: Autocorrelation Function

AIC: Akaike Information Criterion

GPU: Graphical Processing Unit

LSTM: Long Short Term Memory

MASE: Mean Absolute Scaled Error

MIMO: Multi Input Multi Output

MISO: Multi Input Single Output

MSE: Mean Squared Error

RMSE: Root Mean Squared Error

RNN: Recurrent Neural Network

SARIMA: Seasonal Autoregressive Moving Average Model

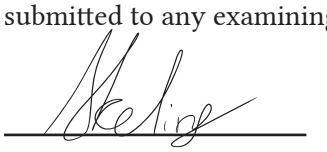
VAR: Vector Autoregression Model

VRAM: Video Random Access Memory

WMO: World Meteorology Organisation

Affidavit declaration

I, Alexander Steding, hereby declare under oath that I have independently and exclusively completed the present bachelor thesis using only the sources listed in the bibliography.The work has not been submitted to any examining authority in this or a similar form before



Hannover, 27. August 2023