Freie Universität Berlin

Bachelorarbeit am Institut für Informatik der Freien Universität Berlin

Arbeitsgruppe Machine Learning and Robotics

Sequence-to-Sequence Models for Pollutant Load Forecasting

David Reuschenberg
Matrikelnummer: 4902427
david.reuschenberg@fu-berlin.de

Betreuer: Thomas Meiers
Eingereicht bei: Prof. Dr. Daniel Göhring
Zweitgutachter: Prof. Dr. Rául Rojas

Berlin, June 24, 2021

Abstract

Air pollution is still a major problem in cities all over the world, therefore a short term forecast of pollutant loads across critical regions is of interest for decision makers to take appropriate measures. In this thesis, a sequence-to-sequence recurrent neural network has been implemented and trained to perform a 48h pollutant load forecast for measuring stations across North-Rine-Westfalia on the basis of measured pollutant load and weather data. The network is based on Gated Recurrent Units (GRU) and extended with an attention mechanism. The network outperforms a baseline neural network and the attention mechanism provides interpretability of the results. The thesis is developed in cooperation with the Fraunhofer Heinrich-Hertz-Institute as part of project SAUBER.

Eidesstattliche Erklärung

Ich versichere hiermit an Eides Statt, dass diese Arbeit von niemand anderem als meiner Person verfasst worden ist. Alle verwendeten Hilfsmittel wie Berichte, Bücher, Internetseiten oder ähnliches sind im Literaturverzeichnis angegeben, Zitate aus fremden Arbeiten sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungskommission vorgelegt und auch nicht veröffentlicht.

June 23, 2021

David Reuschenberg

Contents

1 Introduction

Air pollution leads to numerous dangerous health problems and is still one of the biggest challenges for cities all over the world to overcome. Not only human health, but also the environment is put at risk, as terrestrial and aquatic ecosystems are threatened by air pollution. [10] The impact on human health has of course also economic consequences, such as health care costs, days off through illness or hospital admission. The timeliness of the problem is shown by the fact that only in June 2021, Germany has been judged by the European court for surpassing threshold values for pollutant load concentrations in several cities from 2010-2016. The relevant pollutants in Germany at the moment are nitrogen oxides (NO, NO_2), groundlevel ozone (O_3) and particulate matter (PM). NO and NO_2 pollution is caused mostly by traffic and industry, PM by traffic and heating and O_3 by solar radiation. [10]

For decision makers in cities, a short-term forecast of pollutant load values is of interest for taking appropriate short-term measures such as establishing low emission zones, road cleaning, decreasing speed limits or increasing parking fees. Projekt SAUBER (Satellitenbasiertes System zur Anzeige, Prognose und Simulation von Luftschadstoffen für eine nachhaltige Stadt- und Regionalentwicklung) is funded by the BMVI as part of the research initiative mfund. It aims to develop a application to display a 48 hour forecast of polluant load concentration in the form of a 2D map over Northrhine Westfalia (NRW) and in Stuttgart, Baden-Wuertemberg (BW). Fraunhofers Heinrich-Hertz-Institut (HHI) is one of the project partners in charge of developing a backend deep neural network to perform the forecast and upload it on the partner server. The data set for the project consists of hourly pollutant load measurements from measuring stations across NRW and around Stuttgart and weather data from measuring stations of Deutscher Wetterdienst (DWD). In this thesis, which is developed in cooperation with HHI, a neural network architecture will be designed for the preliminary task of predicting pollutant loads on the locations of the measurement stations themselves. This network will be implemented, theoretically analysed and empirically evaluated.

Recurrent Neural Network (RNN) architectures, being successful in Neural Language Processing (NLP) tasks in the past, have shown promising results also in multi-horizon time-series forcasting problems, where multiple future timesteps have to be predicted on the basis of multiple past timesteps. A RNN-based encoder-decoder model has been designed and implemented for the task at hand.

In section 2, the theoretical foundations of recurrent neural networks and sequence to sequence models are established, relevant literature is reviewed and in section 3. Section 4 details the data engineering and presents the developed architecture and section 5 presents the experiments carried out with that architecture. Section 6 summarizes the results and describes further steps to be carried out in the future development of the project.

1.1 Own contribution

The prediction model was developed in collaborative discussions in our team at HHI. I programmed the whole machine learning part of the project, including model defini-

tion, data loader, training and evaluation. The server infrastructure, data acquisition and data preprocessing has not been implemented by me. The code and experimental results are available on the gitlab instance https://git.imp.fu-berlin.de/davidreusch/sauber_s2s

1.2 Technical Details

All of the programming for the project was done with Python 3.8. The data are stored in a PostGreSQL database and accessed via the python library SQLAlchemy. Data preparation and preprocessing was done using Numpy and Pandas. The neural network, training and prediction was implemented with PyTorch 1.4.

2 Fundamentals

2.1 Recurrent Neural Networks

2.1.1 Vanilla RNN

Recurrent Neural Networks, originally presented by Rumelhart [13], create an internal representation of input data, the so called hidden state, that is used in subsequent computation steps. One natural application for this architecture consists of problems where the input data can be represented as a sequence of input vectors. This sequence can be consumed one token after the other by the RNN, and with every token the RNN produces a hidden state that is consumed in the next timestep together with the next input. In this way, information can flow from earlier to later timesteps, enabling the model to learn dependencies and recurring patterns in the dataset. Given an input sequence $x_1, \dots, x_m$, the RNN can be represented as computing a recursive function f with

$$h_t = f(x_t, h_{t-1})$$

where $x_t \in \mathbb{R}^n$ is the input vector and $h_t \in \mathbb{R}^k$ the hidden state at step t for $1 \leq t \leq m$ and $h_0 = 0$. In the "Vanilla" variant of the RNN, f is a linear projection of x_t and h_t, followed by a non-linear activation function, usually the tanh-function. So the above formula becomes:

$$h_t = \tanh(W^{hx} x_t + U^{hh} h_{t-1} + b)$$

where $W^{hx} \in \mathbb{R}^{k \times n}$ and $U^{hh} \in \mathbb{R}^{k \times k}$ are weight matrices associated to input and hidden state and $b \in \mathbb{R}^k$ is a bias weight. Given target values $y_1, \dots, y_m$ and a loss function $\mathcal{L} : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}^+$, we can compute the empirical loss of a sequence as

$$\sum_{t=1}^{m} \mathcal{L}(h_t, y_t)$$

The expected loss can then be minimized by training the network with the "back-propagation through time"(BPTT) algorithm [12].

2.1.2 LSTM

The Vanilla RNN suffered from the problem of either "exploding" or "vanishing" gradients. A formal analysis of this problem can be found in [8]. Informally the problem can be described as follows: Since the hidden state is multiplied over and over with the same weight matrix in the forward pass over a sequence, in BPTT, we have to take the hidden state weight matrix to the power of as many timesteps, as we want to backpropagate the error. Taking multiple matrix powers leads to an exponential increase of eigenvalues greater than one and to exponential decrease of eigenvalues smaller than one [8], resulting in either very large gradients, leading to instability in learning, or to vanishing gradients, preventing weights from being updated further. This problem was addressed by the LSTM unit [3]. Here, an additional so-called cell-state is introduced, whose value can be controlled via learned gating parameters. The following equations describe the LSTM:

$$
\begin{aligned}
f_t &= \sigma(W_f x_t + U_f h_{t-1} + b_f) \\
i_t &= \sigma(W_i x_t + U_i h_{t-1} + b_i) \\
o_t &= \sigma(W_o x_t + U_o h_{t-1} + b_o) \\
\tilde{c}_t &= \sigma(W_c x_t + U_c h_{t-1} + b_c) \\
c_t &= f_t \circ c_{t-1} + i_t \circ \tilde{c}_t \\
h_t &= o_t \circ \tanh(c_t)
\end{aligned}
$$

where $x_t \in \mathbb{R}^n$ is the input vector and $h_t \in \mathbb{R}^k$ as before, $c_t \in \mathbb{R}^z$ is an additional cell state at time t, σ is the sigmoid function and $\circ$ is the Hadamard product. $W_l \in \mathbb{R}^{z \times n}$, $U_l \in \mathbb{R}^{z \times k}$ and $b_l \in \mathbb{R}^z$ for $l \in \{f, i, c\}$ and $W_o \in \mathbb{R}^{kxn}$, $U_o \in \mathbb{R}^{kxk}$ and $b_o \in \mathbb{R}^k$ are weights assoziated with the input and hidden state.

Here, $\tilde{c}_t$ can be seen as the new candidate cell state computed as a nonlinear function of the last hidden state h_{t-1} and current input x_t. f_t can be then seen as a "forget" gate, that controls, how much of the previous cell state is preserved for the next computation, i_t is an update gate, which determines how much influence the new candidate cell state $\tilde{c}_t$ will have in the new cell state and o_t is an output gate, that controls, how much of the computed cell state is carried over to the new hidden state. Through the gating vectors, the network can for example conserve values from earlier timesteps by setting the respective values in output and forget gate to 1, or forget old values and introduce new values by setting output gate to 1 and forget gate to 0.

2.1.3 GRU

Gated-Recurrent-Unit (GRU) Models [2] can be seen as a simplified version of the LSTM. Here, we have no additional cell state and only two gating vectors computed. GRU can be described by the following equations:

$$
\begin{aligned}
z_t &= \sigma(W_z x_t + U_z h_{t-1} + b_z) \\
r_t &= \sigma(W_r x_t + U_r h_{t-1} + b_r) \\
\tilde{h}_t &= \tanh(W_h x_t + U_h(r_t \odot h_{t-1}) + b_h) \\
h_t &= (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t
\end{aligned}
$$

where $x_t \in \mathbb{R}^n$ and $h_t \in \mathbb{R}^k$ as before, W_z, W_r, $W_h \in \mathbb{R}^{k \times n}$, U_z, U_r, $U_h \in \mathbb{R}^{k \times k}$ and b_z, b_r, $b_h \in \mathbb{R}^k$ are weights assoziated with the input and hidden state.

The rational behind these equations is similar to the case of LSTM, but here the gating is organized differently. Now we have only two gates: A "reset" gate r_t determines, how much the old hidden state h_{t-1} contributes to the new hidden state candidate $\tilde{h}_t$ and the new hidden state h_t is then computed as a weighted sum of the hidden state candidate and the old hidden state weighted by the "forget" gate z_t.

2.1.4 Encoder-Decoder-Scheme

Sequence-to-Sequence (S2S) models were originally developed for language translation tasks ([15], [2]). The model consists of two parts: An encoder network processes an input sequence from the source language and produces a hidden state encoding of this sequence. A decoder network receives this hidden state as an input and produces an output sequence in the target language step by step in an autoregressive loop.

S2S models have shown promising results also in time-series-forecasting problems ([16], [4], [14]). For timeseries-forcasting, we have an input $x_1, \ldots, x_m$, $x_i \in \mathbb{R}^n$ for $i \in \{1, \ldots, m\}$and have to make some predictions $\hat{y}_{m+1}, \ldots, \hat{y}_{m+d}$, $y_i \in \mathbb{R}^z$ for $i \in \{m+1, \ldots, m+d\}$ that will be compared with the observed target series $y_{m+1}, \ldots, y_{m+d}$ of same dimension. By the construction previously described, a RNN encoder can consume the input time steps and produce a representation h_m. The prediction can then be made by training another RNN, the decoder, that receives the hidden state h_m, also called encoding or context vector, as an input, optionally together with an additional input x_{m+1}. The decoder then computes a prediction $\hat{y}_{m+1}$ and a hidden state h_{m+1}, and in the next timestep, it consumes both the prediction and the hidden state. In this way the whole prediction sequence is constructed. So we have the following formula for the decoder:

$$(\hat{y}_t, h_t) = f(x_t, h_{t-1}, \hat{y}_{t-1})$$

for $m < t \leq m + d$ where f is one of the RNN-cells described above. y_m can be the last observation of the values to be predicted, or initialized to 0 if those are not available. If the input $n > 1$, the time series is also called multi variate. Then the input consists of multiple time series and the network can learn relations and dependencies between the single series.

2.1.5 Attention

It can be argued, that in the presented encoder-decoder-architecture, the hidden state that gets passed from the encoder to the decoder represents an information bottleneck, since all the information about the sequence read by the encoder has to be

encoded into a single vector [1]. Furthermore, this vector will be stronger influenced by nearer, i.e. later timesteps, so that earlier time steps have even less influence in the decoding process. In NLP, some word that comes early in the sentence of the source language can influence the translation of a late word in the target language, so this presented a big problem. Bahdanau et. al. [1] first suggested to integrate an attention mechanism to address this problem. The idea is the following: In each timestep of the decoder loop, the decoder computes a new context vector as a weighted sum of the encoder hidden states. The weights are determined by computing some similarity measure between the current decoder hidden state and each encoder hidden state. So the decoder can learn to access relevant information from the encoder hidden states individually for each decoding timestep. So for a decoding timestep $m + i$ the context vector c_i is computed as

$$c_i = \sum_{j=1}^{m} \alpha_{ij} h_j$$

where α_{ij} is the weight of the hidden state h_j, computed as:

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{m} \exp(e_{ik})}$$

This is a softmax function, normalizing the precomputed energies e_{ij}, so that they lie between 0 and 1 and add up to 1. The energies are computed as:

$$e_{ij} = s(h_{m+i}, h_j)$$

where s is some similarity measure of the decoder hidden state h_{m+i} and the encoder hidden state h_j. s can be computed in many different ways, of which the simplest is to let it be the dot product of $\mathbb{R}^n$, but s can also be jointly learned by the network. Luong et. al [7] describe three ways to compute the similarity:

$$s(h_{m+i}, h_j) = \begin{cases} h_i^\top h_j & \textit{dot product} \\ h_i^\top W_a h_j & \textit{general} \\ v_a^\top \tanh(W_a [h_i; h_j]) & \textit{concat} \end{cases}$$

where W_a is a weight matrix and v_a a weight vector jointly trained with the rest of the network. The third variant was the one proposed by Bahdanau. Since Luong preferred the second "general" variant, I will call this "Luong" attention for the rest of this thesis.

This context vector is then used as an additional input for the decoder, so the decoder computes

$$(\hat{y}_t, h_t) = f(x_t, c_t, h_{t-1}, \hat{y}_{t-1})$$

where x, h, $\hat{y}$ as before. The context vector can be used in different ways in the decoder computation. Luong concatenates it with the current decoder hidden state h_t just before the output layer, Bahdanau feeds it in as input to the decoder RNN to compute the current hidden state.

3 Related Work

Sequence-to-Sequence models have been applied to time-series forecasting problems in a a range of publications, of which the following only represent a few.

In [6], the authors present a winning solution for the KDD cup in air quality prediction. Their model consists of an ensemble of three models, a random forest model, a gated DNN and a sequence to sequence model with GRU units. The latter showed already good results alone and this model served as one of the main inspirations for the model presented here.

In [16], the authors develop a sequence to sequence model with a classical attention similar to the one implemented in this work and test it on part of the UCR dataset for time series data.

[4] develop a RNN with a convolutional, a recurrent skip and an attention component. The convolutional component shall extract short term temporal patterns and the attention component can select between the relevant extracted patterns. The skip component shall enable long term information flow through the network.

[14], similarly to [4] presents an RNN with a convolutional component and attention over the output of the convolution. Here, the convolution is over each feature of a window of past hidden states.

In [5], the authors present a RNN encoder decoder model combined with a transformer layer. Transformers are sequence to sequence models based solely one the attention mechanism, first introduced by [17]. This is an interesting approach variable selection and static encoding components are integrated to the model and a quantile forecast is performed.

4 Main

4.1 Data aquisition

Table 1 gives an overview over the input data used to train the model. Historical pollutant load data from 2010 until 2020 have been acquired from 52 measuring stations across North-Rhine-Westfalia (NRW) and 4 measurement stations in Stuttgart. All the stations in NRW have measurements of nitrogen monoxide (NO), nitrogen dioxide (NO_2) and particulate matter (PM_{10}) and 20 stations have measurements of ozone (O_3) in addition. The Stuttgart stations only have consistent measurements of NO_2. For the weather data, data has been collected from stations of Deutscher Wetterdienst (DWD) across NRW and around Stuttgart. The values used for the dataset are temperature, precipitation, sunshine, wind speed, wind direction and amount of rain. A periodic fetcher is downloading data from the NRW-, LUBW- and DWD-servers and storing it in the SAUBER database every day. The input data from preprocessing will be described in the preprocessing section. All data are collected in hourly resolution

and missing data have been linearly interpolated for up to 3 hour wide gaps.

Input parameter	Source	Unit
NO	Lanuv	$\mu g/m$
NO_2	Lanuv	$\mu g/m$
PM10	Lanuv	$\mu g/m$
O_3	Lanuv (20 stations)	$\mu g/m$
Air Temperature	DWD	°C
Precipitation	DWD	mm
Sunshine duration	DWD	hour
Humidity	DWD	%
Wind direction	DWD	sine, cosine
Wind speed	DWD	m/sec
Year	Preprocessing	integer
Month	Preprocessing	sine, cosine
Weekday	Preprocessing	sine, cosine
Hour	Preprocessing	sine, cosine
Workfree Day	Preprocessing	boolean

Table 1: Data overview

4.2 Statistical Analysis

4.2.1 Recurring patterns

The first important fact to observe about the distribution of pollutant load concentration is that it shows recurrent patterns on different levels. Figure 1 shows the hourly average over one year of NO_2 concentrations, contrasting an urban station with a rural station in NRW [9].

For the urban station, we see a clear temporal pattern: Considering a one day period, we have two peaks of pollution load on the weekdays, corresponding to morning and afternoon rush hour. Considering a one week period, we have a lower average pollution load on weekends than on weekdays. For the rural station, by contrast, we have a low pollution value in general, not showing the aforementioned patterns. A forecasting system has to account for the temporal as well as the spatial dimension, recognising where to forecast according to a specific temporal pattern and where not. The PM_{10} concentration shows not such a characteristic daily pattern, but a clear pattern over a year. Since PM_{10} pollution is caused mostly by heating and only secondarily by traffic, it is clear that we have a higher average PM_{10} concentration in the winter, as can be seen in figure 2 [9].

We also have a long-term temporal effect to the extent that pollution load has continuously decreased over the last decades due to more environmentally friendly technologies both in transport and in heating, as can be seen in figure 3 [10]. So the non cyclic progression of time over the years is also an important parameter. Historically unique events like the corona pandemic can also have a large influence

on pollutant load. In figure 4 we see a comparison of the average pollutant load over 2019 and 2020 and we see a clear decline of NO_2 pollution [9]. In reality, of course the time as such plays no causal role, but for a good forecast, these parameters are nevertheless important, since we do not have a physical simulation model and have to make the forecast on the basis of statistical patterns.

4.2.2 Causal relationships

In figure 5, we see the the weekly average O_3 load in the rural station EIFE in contrast to the urban station WALS. Higher O_3 concentrations are caused by higher sun radiation, so both stations have in common a peak around midday for each day. However, at the urban stations, the daily minima are much lower in WALS than in EIFE. This is due to the chemical reaction: $3NO + O_3 = 3NO_2$ [11]. Since NO concentration is much higher in cities than in rural regions, the average O_3 concentration is less there, as can also be seen in figure 6 [9].

4.3 Preprocessing

Since the weather stations are in different locations than the pollutant measurement stations, the weather data at the pollutant stations had to be interpolated. For this, an inverse distance weighting has been applied to the values from weather stations in a 25 km radius around the pollutant load stations in the following way:

Let the weather station locations in the radius of station location x be x_i for $i = 1, 2, \ldots, n$ and the value of feature j $v_j(x)$ for $j = 1, 2, \ldots, m$, then for interpolating the value of the jth feature at location x, we use the following form:

$$v_j(x) = \frac{\sum_{i=1}^{n} w(x, x_i) v_j(x_i)}{\sum_{i=1}^{n} w(x, x_i)},$$

where $d(x, y)$ is the euclidean distance between x and y and $w(x, y) = \frac{1}{d(x,y)}$ for locations x, y. Since no DWD-Station is in the same location as a pollutant measurement station, in which case we could simply take the value of that station, there is no risk of dividing by zero.

As described in the previous section, recurring patterns over a year, month, week or day are important. To provide information about this to the network, cyclical time features were created by encoding the month, day, weekday and hour as sine and cosine values, so that the network can recognize for example the hour 23 being close to the hour 0. The same has been done also for wind direction, since this was given as degree. Since holidays behave like weekdays in terms of the daily pollutant load concentration, a flag value was created stating if the considered day is a holiday or not. The year itself has been added as an integer, because of the described long-term trends in the data.

A mean-normalization has been applied for each feature independently over the whole data from all stations since 2010. So for each feature x_i the empirical mean μ_i and standard deviation σ_i have been calculated and then for the jth sample $x_i^{(j)}$ the

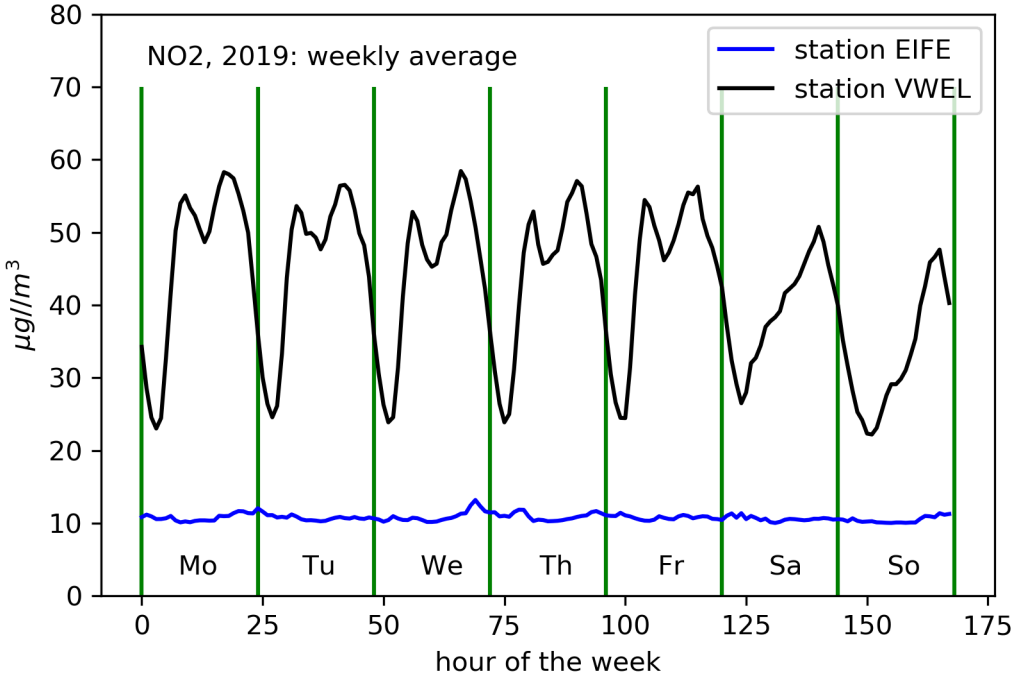


Figure 1: EIFE: Eifel (Simmerath), VWEL: Wuppertal-Gathe

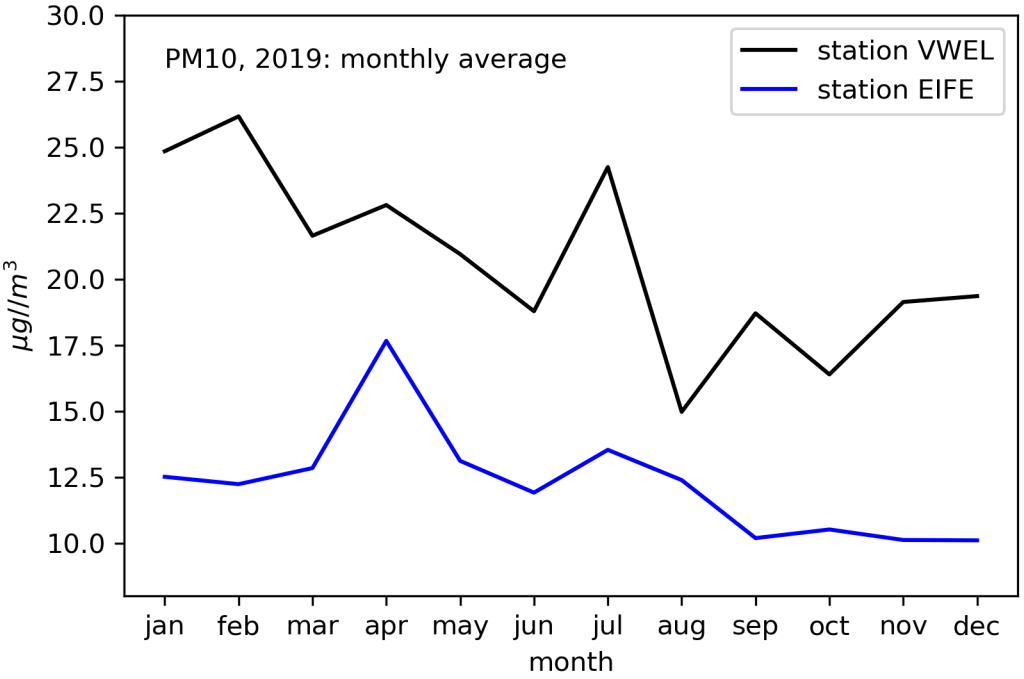


Figure 2: VWEL: Wuppertal Gathe, EIFE: Eifel (Simmerath)

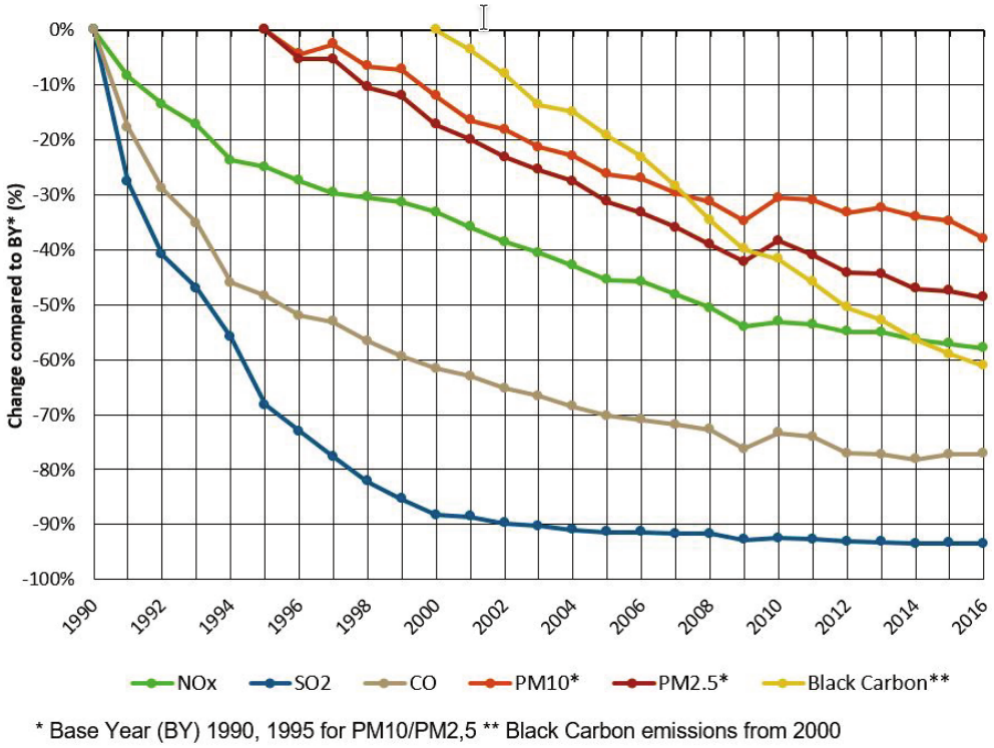


Figure 3

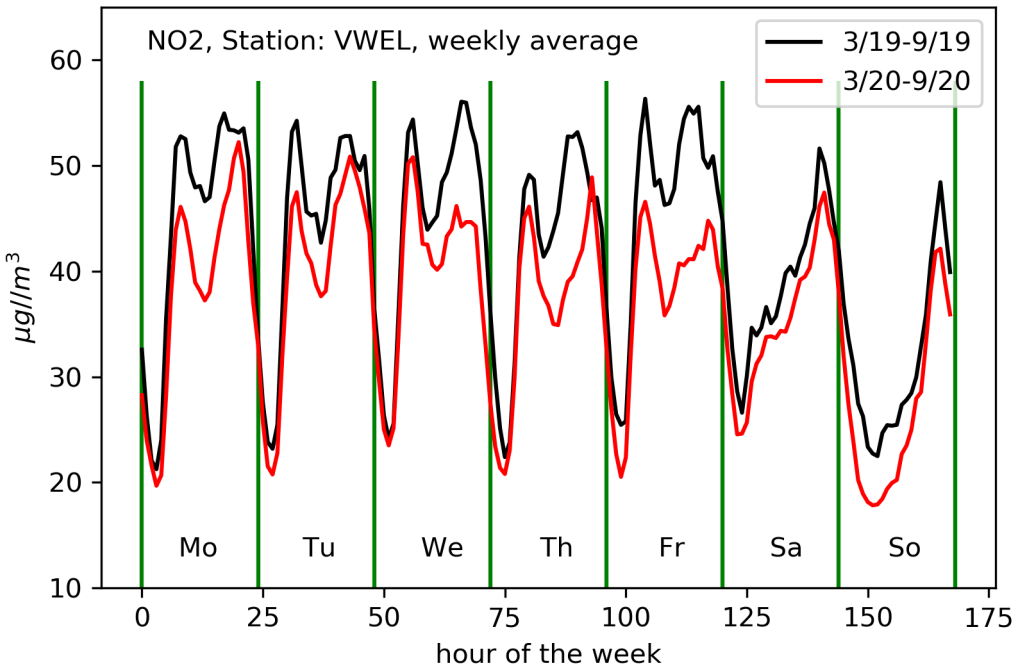


Figure 4: VWEL: Wuppertal-Gathe

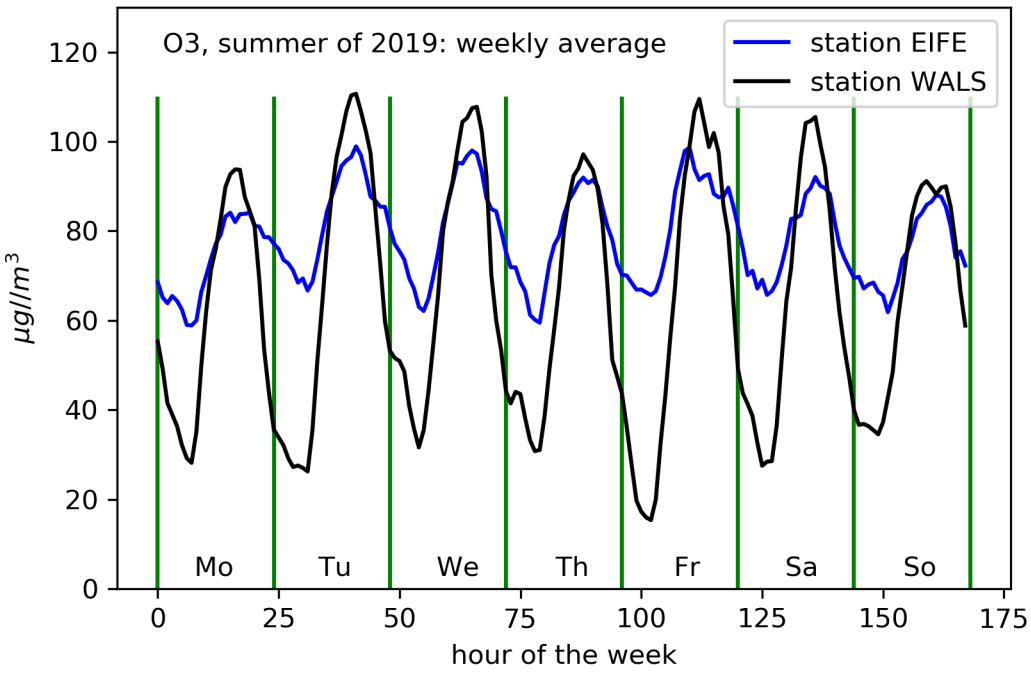


Figure 5: EIFE: Eifel (Simmerath), WALS: Duisburg-Walsum

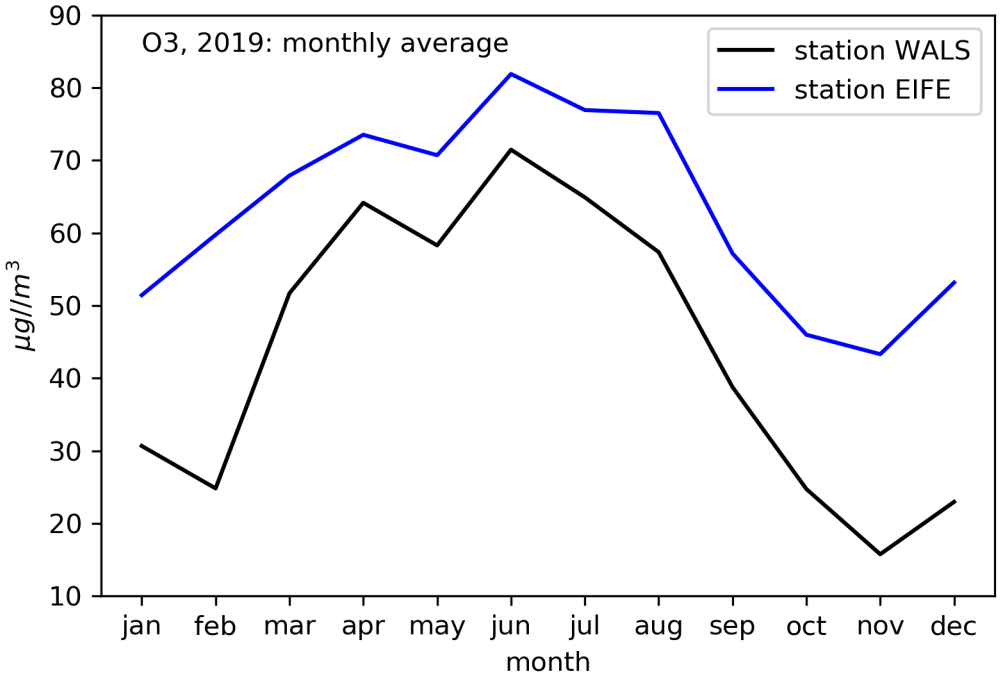


Figure 6: EIFE: Eifel (Simmerath), WALS: Duisburg-Walsum

normalization is computed as

$$\tilde{x}_i^{(j)} = \frac{x_i^{(j)} - \mu_i}{\sigma_i}$$

4.4 Model Architecture

4.4.1 Original Model

The main requirements for the model architecture to be developed were

- maximization of average prediction accuracy in terms of minimizing the average L1 loss of predictions over all the sequences of the validation set. The denormalized L1 loss measure is important because potential system users are interested in whether certain threshhold value for the pollutant concentrations get surpassed, and the L1 loss gives an idea of how capable the network is to make a prediction that is precise enough to decide, if a polluant load concentration will be too high in a certain time period.

- extendability: the architecture should be easily extendable to accept additional sorts of input data. This is particularly relevant for our project, since the input comes from heterogeneous data sources, and additional data sources have to be integrated in multiple development cycles.

A encoder-decoder model (see figure 7) with GRU units has been designed and implemented, following roughly the approach described by [6]. Encoder and decoder are both n-layer GRUs and the decoder has in addition two linear output layers with a rectified linear unit (ReLU) in between. The encoder input x^p consist of all the inputs presented in 1, i.e. the measured pollutant load values and time and weather data of the past week (168 hours), and the decoder input x^f consists only of time and weather data for the next 48 hours.

After having read all the inputs sequentially, the encoder produces n hidden states and the ith layer of the decoder takes the ith produced hidden state as an input hidden state. By letting the decoder have the same layer size as the encoder, all the hidden states can get passed on and no information is lost due to summation or averaging of hidden states. In the first decoding timestep, the decoder receives as input the hidden states from the encoder, the last measured pollutant load values y_0^f, providing the network with a reference start point for the prediction, and future time and weather input data x_1^f. In subsequent timestamps $2 \leq t \leq 48$, the decoder takes its own prediction from the last timestep $\hat{y}_{t-1}$ as input for the prediction of the current timestep t, in addition to the last produced hidden state h_{t-1} and additional input x_t^f. In this way, the whole 48 hour output sequence is produced step by step. For training, a loss function of these predicted values with measured pollutant load values is computed and minimized via batch gradient descent.

The split into an encoder and a decoder provides the model with a natural modularity. Encoder and decoder can have different inputs and can be modified independently from one another. Also additional input data for encoder or decoder can be

fed in flexibly. For example additional static data could be processed by a small additional network and fed in directly to the decoder (see also the outlook on the further project development in section 6).

The model is trained with exact weather data as input for the decoder. For the live prediction, the model will of course only have a weather forecast as weather input, which has itself an error. Historical forecast data were not available, so another possibility to alleviate this issue would have been to add some kind of noise to the exactly measured weather data that we use as an input. Here the problem is, that the noise would have to be from the same distribution as the error of the weather forecast, else a systematic bias would be introduced into the prediction. By taking the exact weather values in training, the prediction accuracy in a live prediction will by contrast simply be proportional to the accuracy of the weather forecast, so the forecast system will profit from improvements of weather forecasting accuracy as well.

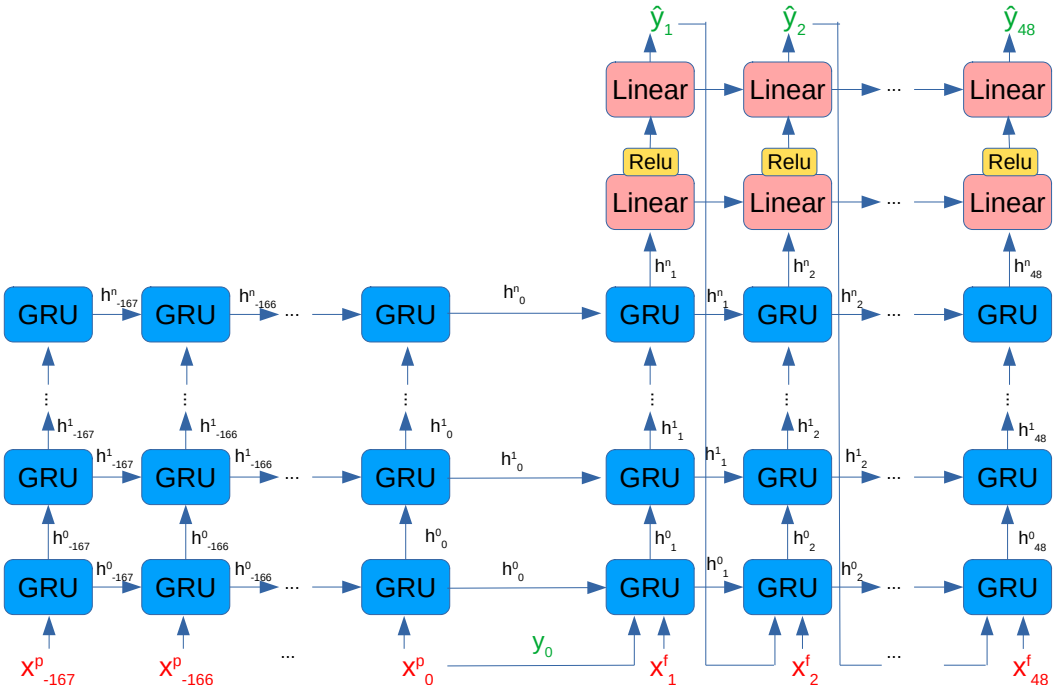


Figure 7: Architecture of the implemented sequence to sequence model. x_p is the input over the past 168 hours, x_f the input for the future 48 hours and y_0 are the last measured polluant load values and $\hat{y}_i$ are the forecasted pollutant values. h_i^k is the hidden state of the k-th layer at timestep i

4.4.2 Extension with Attention

In an attempt to improve on this architecture, an attention mechanism has been integrated, as can be seen in figure 8. The motivation behind using attention is that the network could learn to attend to certain characteristic time steps of the days before to predict the current timestep. The attention mechanism was implemented in the variants mentioned in section 2 (dot product, Luong, concat). Attending over the whole 168 long hidden state sequence of the encoder was unpractically slow due to

the time complexity of the attention algorithm, where $m \times d$ many attention weights have to be computed, when m is the number of hidden states attended over and d the length of the decoder output sequence. Assuming that nearer timesteps have a bigger influence on the decoder prediction, which is confirmed by ablation of the first half of the encoder input (see section 5.3.1), only an end part of the encoder hidden state sequence was attended over and the length of that sequence was treated as a hyper parameter. The context vector is computed from the hidden states of the top layer of the encoder and concatenated with the top hidden state of the decoder, as described in [7]. The concatenated vector then goes into a fully connected layer, whose size is adapted accordingly.

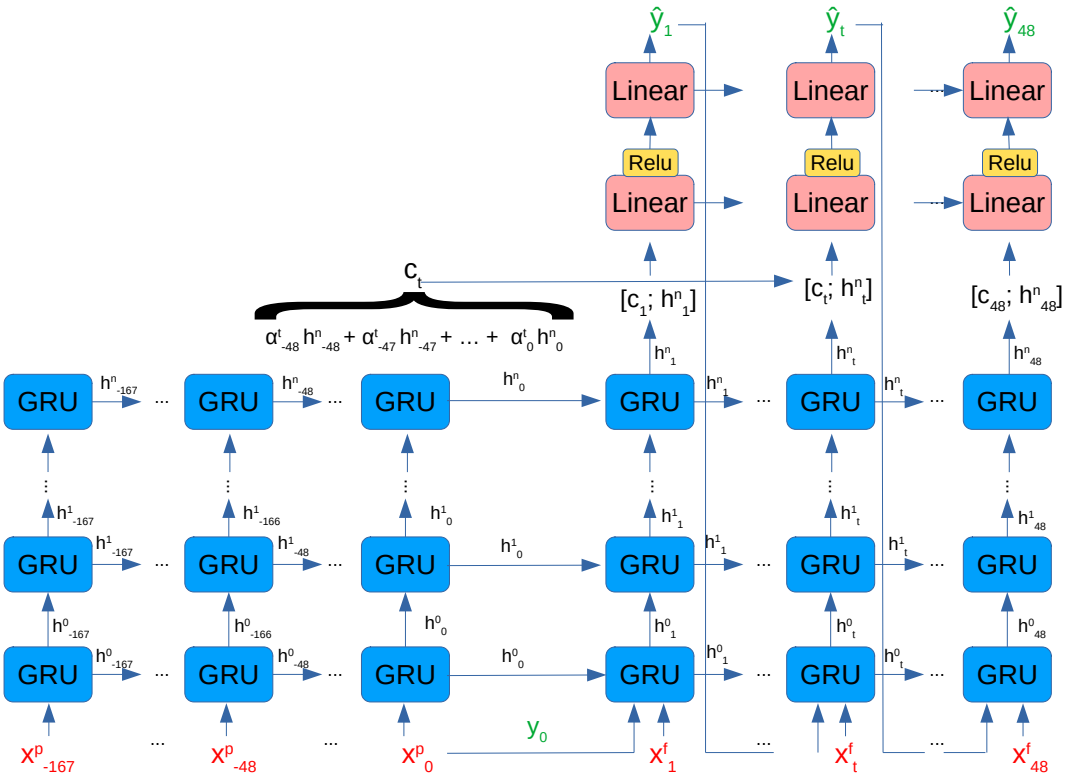


Figure 8: Architecture of the implemented model with attention of length 48. c_t is the context vector for timestep t, $[a; b]$ means concatenation of vector a with b, $\alpha_i^t = \text{softmax}(f(h_{t-1}, h_i))$, where f is one of the mentioned alignment models, other variables as before

5 Experiments

5.1 Training Set Generation

For training, overlapping sequences with stride 1 have been created with 168 hours (one week) of past input data for the encoder, 48 hours of subsequent weather data and time input data for the decoder and 48 hours of corresponding pollutant load

labels. For splitting the data into training, validation and testset, there was a trade-off between the amount of randomization and hence least possible bias in the split, and the amount of training data lost due to the splitting, because the 216 step long sequences have to be taken from continuous time regions and cut off at the borders between regions going to different datasets. We decided to split the data according to months, choosing for each station for each year one month at random going to validation and one month going to testset, and the rest going to training set. So the training, validation and test set ratio is about 10/12 to 1/12 to 1/12. For all stations of NRW this added up to 2.299.349 samples in train, 169.523 in validation and 168.183 in test sets. These sequences where shuffled and batched together with a batchsize of around 1024. Here we can only give an approximate value because the batch size is constrained by available GPU memory, so the larger the hidden state size, the smaller the batch size has to be.

5.2 Training

Three separate models have been trained: One on data from all the NRW-stations producing a forecast of NO, NO_2 and PM_{10}, one on data from only the NRW stations where O_3 measurements were available, producing a forecast of O_3 in addition to the other values and one model for the 4 stations in Stuttgart, producing a forecast of NO_2. The latter will not be further analysed in this thesis.

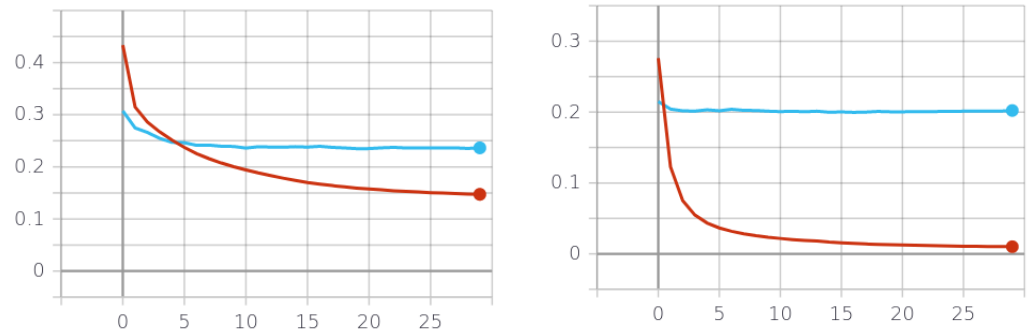
Different combinations of hyperparameters have been explored, as shown in table 2. LSTM and GRU showed similar results, but LSTM has more parameters and is hence slower to train, so GRU was chosen. Increasing the number of hidden units boosted performance up to the value of 1024, where it reached a plateau. Increasing the number of layers above two, by contrast, did not change performance significantly. The network was trained with L2-Lossfunction and the AdamW-optimizer. With a hidden state size of 1024 it has 20.070.404 trainable parameters in total. Training the network on the complete dataset, which amounts to 77GB of data, using a NVIDIA Tesla V100 GPU with 16GB RAM, took about 35 hours.

Parameter	Value Range	Chosen Value
Hidden State Size	64 - 1280	1024
RNN Cell Type	LSTM, GRU	GRU
RNN layers	1-6	2
Dropout rate	0.1 - 0.7	0.3
Learning rate	0.0001 - 0.01	0.0005

Table 2: Explored and optimized hyper parameters

5.3 Results

The most important evaluation metric to compare the models has been the mean absolute prediction error (L1) on the test data set. Other metrics computed where root mean square error (RMSE) and mean directional loss (MDL). MDL was computed as follows: Let $y_1, \dots y_{48}$ be measured the sequence of pollutant load values for the 48



(a) 128 hidden states (b) 1024 hidden states

Figure 9: L2 loss (vertical axis) in training epochs (horizontal axis), train red, validation blue

hour forecast objective and $\hat{y}_1, \ldots, \hat{y}_{48}$ the prediction of the network. Then the MDL is computed as

$$\frac{1}{48} \sum_{t=2}^{48} \|(y_t - y_{t-1}) - (\hat{y}_t - y_{t-1})\|$$

This loss function measures, how well the network predicts the slope of the measured pollutant load curve.

Also the produced curves were evaluated visually in how good they appeared to capture the trend of the data. As a baseline model, a simple feed forward neural network with three fully connected layers and ReLU activations in between with 33.300.624 trainable parameters has been trained for 50 epochs. Table 3 shows a comparison of denormalized l1 loss, RMSE and MDA for the feed forward and the GRU architecture for the ozone stations and table 4 shows this comparison on the complete dataset without ozone. It is apparent, that the feed forward network generalizes much worse than the S2S model, and even more so for the bigger dataset.

9 shows the L2 training loss vs. validation loss for a network with 128 hidden states and one with 1024 hidden states on the ozone dataset. For the large network, validation loss stagnates quickly after a few training epochs and training loss decreases quickly to near 0. The smaller network has a better proportion of training and validation loss, but also here validation loss stagnates after a few more epochs. Since the validation loss does not increase, the network does not overfit, but it also does not generalize the learning during training to a great extent. Increasing dropout above 0.3 resulted in a better proportion of train and validation loss, but decreased validation loss in total. Maybe the fact that we have a strong overlap in the training sequences could be a reason for the stagnation of validation loss, because the network sees a lot of similar sequences in training so that it gets easier to predict the training sequences.

5.3.1 Ablations

Table 5 and 6 show the results of several ablation experiments that were conducted to estimate the effectiveness of parts of the network configuration. Cutting out the whole encoder part of the network, initializing the hidden states of the decoder with zeros,

leads to still surprisingly good results. This could for example mean, that the information transfer between encoder and decoder over the single stack of hidden states is not efficient enough to convey the relevant information from the past sequence to the decoder. Or the history of a specific sequence is not so important for the actual prediction of that sequence and future weather and time data with a starting value of pollutant measurement are mostly sufficient. Both these reasons are also consistent with the results of the next experiment, where the length of the encoder input was halved. Here we only have a slight decrease in accuracy. Investigating the first hypothesis, an attention mechanism was implemented to improvethe information flow from encoder to decoder, see the next section 5.3.2.

The ablation of the weather data from both encoder and decoder input leads to a significant decrease in precision, showing the effectiveness of those input data. Lastly, to investigate the hypothesis, that the network could learn chemical dependencies between the pollutants and use knowledge about one pollutant to predict the other, ozone was removed from the encoder input and targets of the ozone dataset, letting the network only predict the other pollutants on that dataset. The accuracy decreases indeed, but only slightly. This suggests that maybe such a cross dependency is learned but not accorded a lot of weight for the final prediction.

5.3.2 Attention

Since the ablation results suggested that the encoder part of the network was not so effective, it was natural to search for some mean to improve the information transfer between encoder and decoder. Here, the attention concept was an obvious candidate. The attention algorithm had to be tuned for the length of attended hidden states and the algorithm used. The concat variant was discarded in preliminary tests on a smaller debug datasets because it is slower to train due to the weight computation network, but did not show better results, so the Luong and the dot product variant were tested on the ozone dataset. The best variant was a dot product attention over a window of the 24 last hidden states, very similar to the Luong variant with attention length 24, see table 5.

The visualization 10 of the learned attention weights for each decoder timestep averaged over an epoch confirms the hypothesis stated above. The bright diagonals indicate that a hidden state receives a higher weight on average, if its associated timestep has a difference of around a multiple of 24 hours to the current decoding timestep, including 0. This means that the network learns to attend to hidden states from the same hour of earlier days to predict the current hour. Also the last hidden state of the encoder series receives a higher weight. With increasing number of hidden states the pattern is less pronounced, but the general structure is preserved.

This is an interesting result, that shows the importance of daily periodic patterns suggested by the statistical analysis of the data set. Daily patterns like for example NO_2 peaks at morning and afternoon rush hour or an ozone peak at noon can be learned by the network and be used to make the prediction. It is nevertheless surprising, that the result shows such a clear pattern. Thereby, the attention mechanism provides the model with a natural explainability.

That the improvement of attention is nevertheless small, could have different rea-

Model	L1	RMSE	MDA
FFOzone	4.99	9.21	4.92
S2SOzone	4.25	8.54	4.18

Table 3: Comparison of feedforward and S2S network on ozone test set, losses denormalized to $\mu g/m$

Model	L1	RMSE	MDA
FFnoOzone	6.88	14.74	6.78
S2SnoOzone	5.28	12.75	5.19

Table 4: Comparison of feedforward and S2S network on whole test set, losses denormalized to $\mu g/m$

Pollutant	Full model	Half encoder	No encoder	No weather data
NO	3.3	3.31	3.89	4.09
NO_2	4.6	4.8	6.01	6.33
PM_{10}	2.2	2.25	2.59	2.96
O_3	6.91	6.86	8.15	10.99
total	4.25	4.31	5.16	6.1

Table 5: L1 Loss ($\mu g/m$), ablation experiments on ozone test set

Pollutant	Full model	No encoder	No weather data
NO	7.61	9.19	10.27
NO_2	5.77	6.65	8.05
PM_{10}	2.46	2.41	3.22
total	5.28	6.08	7.18

Table 6: L1 Loss ($\mu g/m$), ablation experiments on whole test set

Pollutant	With Ozone	Without ozone
NO	3.3	3.28
NO_2	4.6	4.79
PM_{10}	2.2	2.3
total	3.37	3.46

Table 7: L1 Loss ($\mu g/m$), model with ozone and without ozone input on ozone test set

Pollutant	No attention	Luong 24h	dot product 24h
NO	3.3	3.23	3.27
NO_2	4.6	4.63	4.57
PM_{10}	2.2	2.22	2.22
O_3	6.91	6.67	6.64
total	4.25	4.19	4.18

Table 8: L1 Loss ($\mu g/m$), model with and without attention on ozone test set

sons. Probably without attention, the network can already learn that the values from the same daytime are important and can identify these values, because we pass the hour as a parameter to the model. Information about these values can be passed to the decoder through the last hidden states. Attention would then support a mechanism that is already working reasonably well. This hypothesis is coherent with the fact that the attention weights show a higher variance around the 24 hour diagonals with increasing hidden state size. The bigger the hidden state passed to the decoder, the less attention is needed to extract the relevant information, whereas with a smaller hidden state size the network relies more on attention. The reason could also have to do with the implementation of the attention part and its interaction with the rest of the network. Here, further experiments and more hyper parameter tuning would have to be carried out to maybe find some more effective way to integrate the attention mechanism. Especially the processing of the context vector in the decoder can be done in different ways.

Since the attention shows such a clear pattern, a possible improvement would be to preselect the hidden states that lie around a multiple of 24 timesteps in the past, so that for decoding timestep i we would choose the hidden states attended over as $H = \{h_{i-24k+j} | k \in \{0, \dots, 7\}, j \in \{-d, \dots, d\}, d \in \mathbb{N}\}$, where d would be some chosen margin value. This would make the attention quicker to run and it would be feasible to attend over vectors from a broader time range.

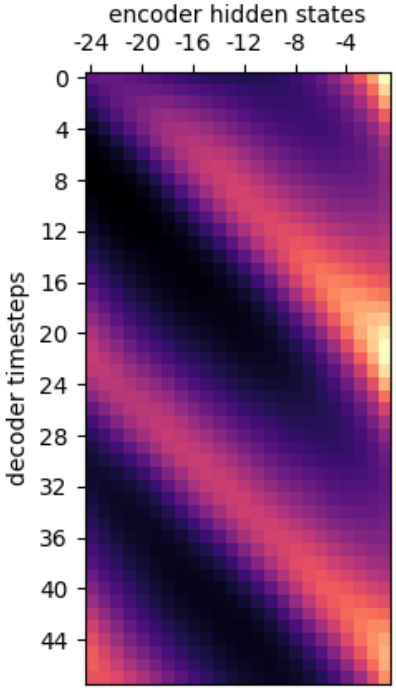
5.3.3 Plots

Figure 11 shows some examples of predictions on the validation set in contrast to the real measured values (note that the scales are different from figure to figure). In general, the network shows a high accuracy but has the tendency to smoothen the results in the prediction. Positive and negative peaks in the curves are mostly detected by the network, whereas fluctuations with lower amplitude are smoothened. Figure a) shows an example of a PM_{10} prediction where the measured values were below the minimum detection threshold for PM_{10} of $10\mu g/m3$. Here, the network predicts a smooth curve but stays close to the constant minimum. To get an impression of the prediction quality, many more randomly chosen plots of predictions on the validation set are available in the gitlab repository https://git.imp.fu-berlin.de/davidreusch/sauber_s2s.

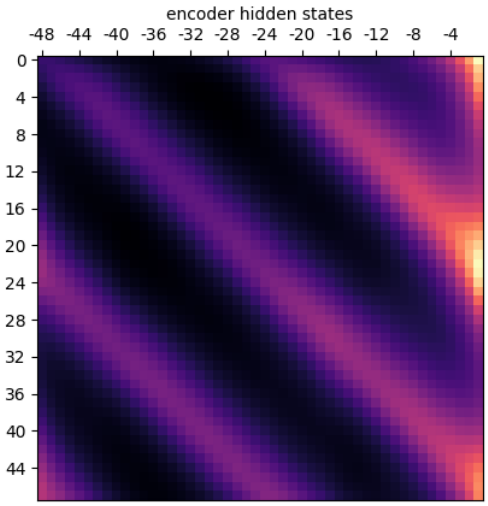
6 Conclusion and Outlook

In this thesis, a sequence-to-sequence model has been succesfully trained for the pollutant load forecasting task at the location of pollutant measurement stations. The loss scores show a high accuracy of the forecast, which is confirmed by validation plots. Ablations show the effectiveness of the weather input and the relatively poor efficiency of the encoder, which is improved by the extension with attention. A visualization of attention weights makes the model interpretable to the extent that a clear focus on timesteps of the same daytime of previous days is visible.

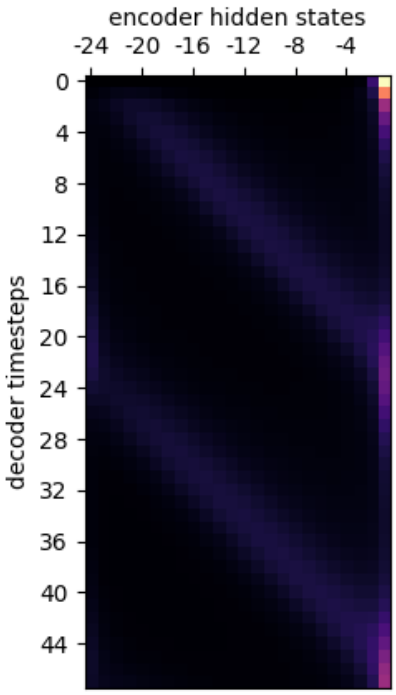
Further research can be conducted in different directions. First, the model architecture can be further improved by refining and tuning the attention mechanism to the



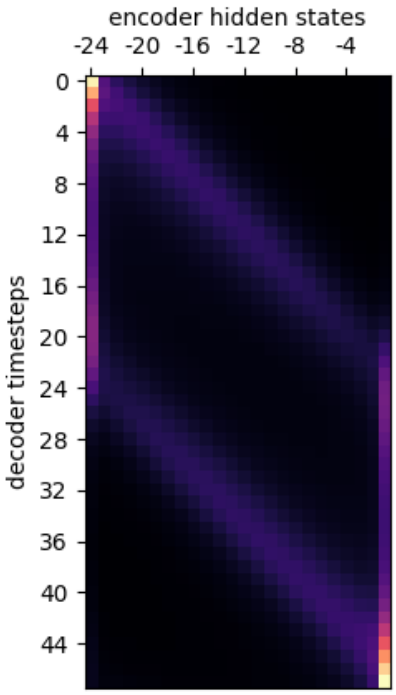
(a) Luong, 128 hidden states, 24h attention



(b) Luong, 128 hidden states, 48h attention

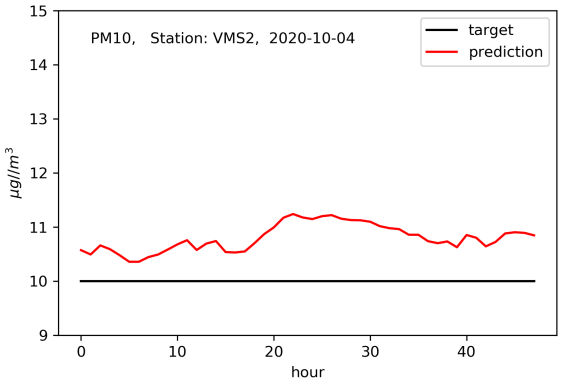


(c) Dot product, 1024 hidden states, 24h attention

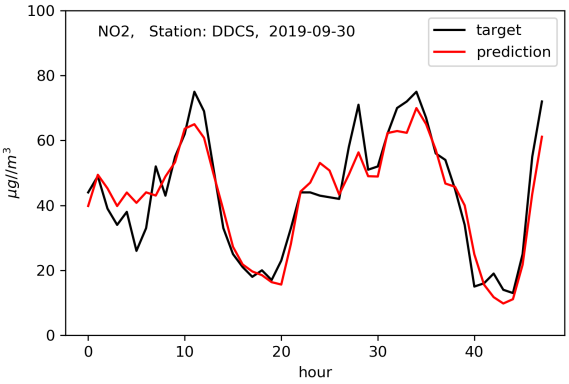


(d) Luong, 1024 hidden states, 24h attention

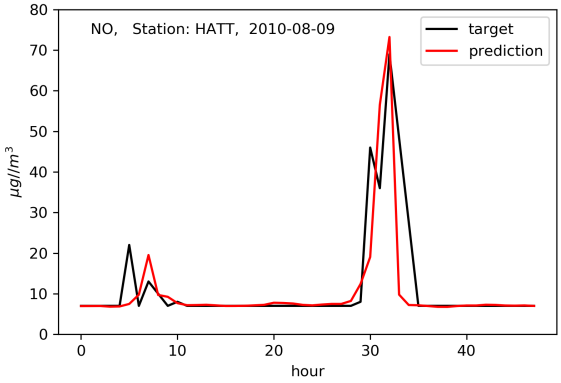
Figure 10: Attention plots, weights averaged over one training epoch on ozone dataset, colormap from low values (left) to high values (right)



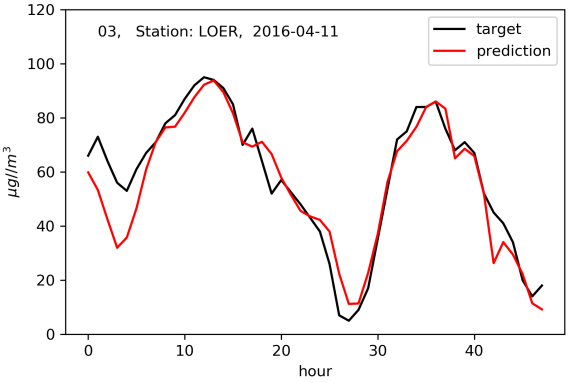
L1 loss: 0.79 $\mu g/m$



L1 loss: 5.22 $\mu g/m$



L1 loss: 5.22 $\mu g/m$



L1 loss: 6.04 $\mu g/m$

Figure 11: Validation plots ozone model with optimized hyper parameters, model prediction vs. ground truth on validation set

S2S architecture. Since the attention mechanism showed the importance of timesteps of the same daytime, these hidden states can be manually chosen as described in the previous section and weighted by the attention mechanism. Also a feature extraction via a convolution over a window of hidden states as described in [14] and [4] could improve the performance.

Secondly, the model can be improved by extending the data sources, which is being done in the next iteration of project SAUBER.

6.1 Further Project Development

In the context of project SAUBER, the model is currently being adapted to do a forecast on arbitrary locations across the considered regions. For this, the network is retrained on the station locations, but this time the pollutant load values are removed from the encoder input and the following additional kinds of data are added:

- Static topographical data such as landscape classification and settlement density: By providing a map with a certain resolution of such data around a forecast location to the network, the network can probably learn relationsships to the pollutant loads at the considered location.
- Hourly traffic density data: Since traffic is directly responsible for NO and NO_2 pollution, the accuracy of the prediction of these pollutants could improve by providing car counts for streets around a considered forecast location
- Satellite data: The european Sentinel S5P satellite provides daily measurements of atmospheric NO_2 and O_3, that can be used to further improve the forecast accuracy.

The measurement stations provide real pollutant load values as labels for the training. Since the topographical, satellite, traffic and weather data are available with a certain resolution all over the considered regions, the network can then perform a forecast with this resolution anywhere in those regions.

References

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[2] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078, 2014.

[3] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.

[4] Guokun Lai, Wei-Cheng Chang, Yiming Yang, and Hanxiao Liu. Modeling long- and short-term temporal patterns with deep neural networks. *CoRR*, abs/1703.07015, 2017.

[5] Bryan Lim, Sercan O. Arik, Nicolas Loeff, and Tomas Pfister. Temporal fusion transformers for interpretable multi-horizon time series forecasting, 2020.

[6] Zhipeng Luo, Jianqiang Huang, Ke Hu, Xue Li, and Peng Zhang. Accuair: Winning solution to air quality prediction for kdd cup 2018. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery amp; Data Mining*, KDD '19, page 1842–1850, New York, NY, USA, 2019. Association for Computing Machinery.

[7] Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation. *CoRR*, abs/1508.04025, 2015.

[8] Razvan Pascanu, Tomás Mikolov, and Yoshua Bengio. Understanding the exploding gradient problem. *CoRR*, abs/1211.5063, 2012.

[9] L. Petry, H. Herold, G. Meinel, T. Meiers, D. Reuschenberg, S. Mirzavand Borujeni, J. Arndt, L. Odenthal, I. Müller, E. Kalusche, T. Erbertseder, H. Taubenböck, B. Weber, S. Jäger, C. Mayer, and C. Gengenbach. Design and results of an ai-based forecasting of air pollutants for smart cities. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, submitted.

[10] L. Petry, Hendrik Herold, Gotthard Meinel, T. Meiers, Inken Müller, E. Kalusche, T. Erbertseder, Hannes Taubenböck, Elaine Zaunseder, V. Srinivasan, A. Osman, Beatrix Weber, Stefan Jäger, C. Mayer, and C. Gengenbach. Air quality monitoring and data management in germany – status quo and suggestions for improvement. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XLIV-4/W2-2020:37–43, 09 2020.

[11] D. Pixteren, S. van Düsing, A. Wiedensohler, and H. Hermann. *Meteorologische Einflüsse auf Stickstoffdioxid*. Sächsisches Landesamt für Umwelt, Landwirtschaft und Geologie, Dresden, 2020.

[12] A. J. Robinson and Frank Fallside. The utility driven dynamic error propagation network. Technical Report CUED/F-INFENG/TR.1, Engineering Department, Cambridge University, Cambridge, UK, 1987.

[13] D. Rumelhart, Geoffrey E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

[14] Shun-Yao Shih, Fan-Keng Sun, and Hung-Yi Lee. Temporal pattern attention for multivariate time series forecasting. *CoRR*, abs/1809.04206, 2018.

[15] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. *CoRR*, abs/1409.3215, 2014.

[16] Yujin Tang, Jianfeng Xu, Kazunori Matsumoto, and Chihiro Ono. Sequence-to-sequence model with attention for time series classification. pages 503–510, 12 2016.

[17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.