Mid-term air quality forecasts using remote sensing data and machine learning

Boris van Linschoten

Master thesis - Computer Science

November 2017

University of Amsterdam	Vrij Universiteit Amsterdam
10176268	2583316

Supervisors:

UvA: Adam Belloum
Rob van Nieuwpoort

Airbus Defence & Space NL: Sjaak Koot

1 Introduction **5**

2 Background **8**
2.1 Air quality forecasting 8
2.2 Time series forecasting using regression 9
2.3 Machine Learning 10
2.3.1 Supervised versus unsupervised learning 11
2.3.2 Training, validation and test dataset 12
2.4 Deep Learning 12
2.4.1 Neural networks 14
2.4.2 Training 15
2.4.2.1 Optimization algorithms 16
2.4.2.2 Epochs 16
2.5 Time series forecasting using Artificial Neural Networks 16
2.5.1 Recurrent Neural Networks 17
2.6 Distributed Deep Learning 18
2.6.1 Model and Data parallelism 18
2.6.2 DistBelief and Downpour SGD 19
2.6.3 Distributed Deep Learning with Apache Spark 20

3 Data **22**
3.1 NO2 22
3.2 Data retrieval 23
3.2.1 Swaths 23
3.2.2 Product levels 25
3.3 Uncertainty 26
3.4 Data flagging 27
3.4.1 Row anomalies 27
3.4.2 Cloud fraction and surface albedo 27

4 Implementation **28**
4.1 Regridding 28
4.1.1 Method 28
4.1.1.1 Filtering 28
4.1.1.2 Algorithm 28
4.1.1.3 Cube data structure 30
4.1.2 Results 30
4.2 Air quality forecasting 32
4.2.1 Base model 35
4.2.1.1 Method 35

4.2.1.2 Results 36
4.2.2 Seasonal ARIMA 40
4.2.2.1 Method 40
4.2.2.2 Results 41
4.2.3 Neural network 45
4.2.3.1 Method 45
Network structure 46
Optimization algorithm 46
Epochs 46
Rolling forecast 47
4.2.3.2 Results 47
Hyperparameters 47
Model results 48
4.2.4 Evaluation 51

5 Conclusion **54**

6 Bibliography **56**

7 Experimentation **59**
7.1 Source detection 59
7.1.1 Peak finding 59
7.1.2 Including wind data 60
7.1.3 Number of observations 61
7.2 Change detection 62
7.2.1 Momentary changes 62
7.2.2 Long-term 64

Chapter 1

1 Introduction

Airbus Defence and Space Netherlands (ADSN) is a company situated in Leiden that provides products for the international aerospace industry, such as instruments, systems and services and solar arrays. One of the instruments developed by the company is the Ozone Monitoring Instrument (OMI). This instrument is part of the EOS-AURA satellite, built under the responsibility of NASA and launched in 2003. OMI collects data not only on ozone but also on various other trace gases in the atmosphere, such as NO2 and SO2, providing daily global coverage with a resolution of approximately 13 x 24km[1]. The TROPOspheric Monitoring Instrument (TROPOMI), the successor to OMI has been launched on October 13th, 2017. With 7x7km, TROPOMI has a much higher resolution than OMI, greatly increasing the amount of data that has to be handled.

Recently, ADSN started developing the 'Atmospheric Composition Services' (ACS) platform. This platform, consisting of an IT-infrastructure, software and processes, will be used to offer commercial services in the domain of atmosphere monitoring, using the data from OMI, TROPOMI and other sources. These services require large amounts of remote sensing data. To illustrate, the OMI data since 2004 for only the gas NO2 is 1.2 terabytes. This size keeps increasing, as new data comes in every day. With TROPOMI, the data volume is expected to grow at even larger rates because of the higher resolution of the instrument. Because of the size of the remote sensing data, conventional methods for both storage and processing will not provide the desired functionality for the ACS platform. This brings up the question where and in what ways Big Data systems can be used in the design of the ACS platform.

[1] Resolution varies because of the earth's curvature. In the center of the satellite's view the resolution will be 13x24km, at the edges the resolution will be lower (i.e. bigger pixels).

The development of the ACS platform is in an early stage, as ADSN is still in the process of researching what commercial services they can offer and for what services there is commercial demand. Some use cases for the platform have been proposed, namely:

- Regridding; the regridding of the irregular remote sensing data to regular, daily grids
- Forecasts; making predictions on atmospheric composition
- Change Detection; detect significant changes in the atmospheric composition
- Source Detection; detect and identify emission sources

In this research, the main research question is on the second use case, making air quality predictions. Towards this goal, regridding is a required preprocessing step and thus this use case will also be implemented.

Primary research question:

➔ *How can machine learning be used to make mid-term air quality forecasts?*

Secondary research questions:

➔ *How can regridding be implemented in a scalable way using Apache Spark?*

➔ *Can Deep Learning improve one-year forecasts compared to simpler models?*

The primary research question focuses on making forecasts up to one year in the future, as opposed to previous studies that have mainly focused on short-term forecasts of a few days in the future. Short-term air quality forecasts are traditionally done using atmospheric diffusion models, that use many different sources to make predictions, such as meteorological data and data on emission sources. However, this data is unavailable or inaccurate for more than a few days in the future. This has prompted us to research the feasibility of making predictions using only the available satellite data and Machine Learning.

The approach towards answering the research questions shares many properties with other possible applications of the ACS platform. As previously mentioned, the applications of this platform require large amounts of remote sensing data together with other analysis tools. In this research we will combine the big data processing engine Apache Spark with Deep

Learning and other libraries. This contributes a proof of concept of efficient analysis of large amounts of geospatial data using a combination of existing methods.

On top of the challenges above, the large amounts of data and the efficient combination of different methods for our analysis, four other main challenges exist in predicting air quality. Firstly, the **processes that determine air quality are complex**. Secondly, these **processes occur at many different scales**. For example, both wind (large scale) and chemical reactions (small scale) are processes that impact air quality in a certain area. Although the ability to predict air quality has improved due to advancements in measuring and modelling atmospheric chemistry, modelling these complex processes operating at different scales is still challenging.

Data availability and **data quality** are the final two challenges in predicting air quality. The available air quality data is either accurate, but not available at a global level or has large inaccuracies. Accurate data is for example collected with local measuring stations, but getting global coverage using this method is not feasible. Data on a global level can be collected through remote sensing, but atmospheric chemistry data from remote sensing has large uncertainties.

The primary research question is addressed by testing and comparing three different methods for making atmospheric composition forecasts. First, a simple 'base model' is constructed based on the data characteristics. Then, this base model is compared with an ARIMA model and a Recurrent Neural Net (RNN).

The first secondary research question adresses the first use case, regridding. This use case is implemented first, because the data we get from the regridding is required for the next steps. In this research a full, scalable regridding solution is implemented using Apache Spark. The desired processing time of under 1 minute per day was achieved using 32 cores. The resulting grids are saved in a Cube data structure.

In the next chapter, the theoretical background for this research will be discussed. The following chapter discusses the data, both the retrieval of the data and its characteristics. Chapter 4 will cover the implementation of the two use cases, the regridding and air quality forecasting respectively. In this chapter both the methods and results for these use cases will be discussed. Finally, in chapter 5 the conclusions of this research will be layed out. The appendix contains two use cases for which experiments were done in the context of this research.

Chapter 2

2 Background

In this chapter, the background for this research will be discussed. First, an overview will be given on related work on forecasting air quality, after which the technical background on forecasting time series in general will be discussed.

2.1 Air quality forecasting

In recent years, forecasting air quality has become a major topic in air quality research due to the known negative health effects caused by airborne pollutants. Air quality forecasts could be used to take preventive and evasive action in case of severe pollution. In this way, by influencing people's daily habits or by placing restrictions on traffic and industry it should be possible to avoid excessive medication, reduce the need for hospital treatment and even prevent premature deaths. This is especially essential where certain sensitive groups in the population are concerned, such as children, asthmatics and elderly people.

Generally speaking, there are two approaches to making air quality predictions. The first approach is to use detailed atmospheric diffusion models, such as the chemical transport model LOTOS-EUROS[2]. This model uses meteorological and emission data in combination with the current air quality state to model the air quality for a few days in the future. A downside to this approach is that it depends on detailed data on emission sources, which is often not available or not reliable. Furthermore, depending on meteorological forecasts, this approach can only make short-term air quality forecasts.

The second approach is to use statistical methods, which attempt to determine the underlying relationship between a set of input data and the target variable. An example of such a method is the linear regression model, which Shi and Harrison (1997) used to predict

[2] http://www.lotos-euros.nl/

hourly NO_2 concentrations in central London [1]. However, one of the limitations imposed by linear regression models is that they will underperform when used to model non-linear systems, such as air quality. Neural networks on the other hand are able to model non-linear systems and may thus perform better. Comrie (1997) compared neural network techniques with regression models for daily ozone predictions, and found that neural network techniques performed consistently better than regression models, although the gains were small [2]. Gardner and Dorling (1998) found similar results when they used neural networks to model and predict hourly NO_x and NO_2 concentrations from readily observable local meteorological data [3]. They stated that the neural network models perform well when compared to previous attempts to model the same pollutants using regression based models.

Where previous research focuses mainly on short-term air quality forecasts, this research focuses on mid-term air quality predictions. This is a time series forecasting problem; the historical remote sensing data since 2005 will be used to calculate the expected air quality several months to one year in the future. To this end, several methods will be applied and their performance compared. First, a simple regression as well as an ARIMA model is used, discussed in section 2.2. Next, we explore if we can improve on these baseline performance using neural networks. Therefore in section 2.3 and 2.4, some background on Machine Learning and Deep Learning is given, after which in section 2.5 the usage of neural networks for time series forecasting is discussed.

2.2 Time series forecasting using regression

Time series are different from other datasets in that they add a time dimension: the observations have an explicit order. This additional dimension is both a constraint and an additional source of information. Time series forecasting uses the information in a time series to forecast future values of the series. Often, the approach is to fit a model on the historical data and use that model to predict future observations. However, many different models can be used. In this section, classical time series forecasting methods will be discussed briefly. Then, after discussing Machine Learning and Deep Learning in general, we will discuss time series forecasting using Deep Learning in section 2.5.

Classical methods for time series forecasting using regression are still very useful as they are established and can be used as a benchmark to test other, alternative approaches against.

The simplest method is to use a simple regression to fit a curve to the historical data. For example, a simple linear regression can already give insight into the linear trend of the data. More complex curves can further improve the model by including more components based on knowledge of the data, such as a sine component to enable the model to capture seasonal components of the data (Brockwell & Davis, 2016) [15]. After such a curve is fit, a forecast can be obtained by extrapolation of the curve.

Probably the most used statistical method for time series forecasting is the AutoRegressive Integrated Moving Average model (ARIMA) [10]. This model provides a powerful method for making time series forecasts because it explicitly caters to characteristics that are often seen in time series data. These key aspects are described in the model's name:

- *AutoRegressive (AR)* means that the model uses the dependency between an observation and a number of lagged observations (i.e. some observations directly before the observation).
- *Integrated (I)* means that the model makes the time series stationary by subtracting an observation at the previous time step from an observation. A time series is stationary if the mean, variance and autocorrelation do not change over time.
- *Moving Average (MA)* means the model uses the dependency between observations and a moving average.

An extension of the ARIMA model, the Seasonal ARIMA model can also deal with seasonality in time series [10].

2.3 Machine Learning

Machine learning gives computers the ability to act *from experience*, instead of being *explicitly programmed.* The core concept of machine learning is using some input data to train a model, then use this trained model to make predictions on new, unseen data. The

training of this model can be seen as the model learning from the data, hence the name machine learning. In this learning process, the model is given new input data step by step and with each step it makes a prediction on the output. Then this predicted output is compared to the real output and the model corrects its parameters based on the error it made in the prediction. These steps are repeated until the model's predictions do not improve anymore.

The idea of machine learning has been around for a long time, with the researcher Arthur Samuel coining the term as early as 1959 [11]. However, only in recent years machine learning has become very popular. Jeffrey Dean[3], the researcher at Google who introduced MapReduce and co-authored the papers introducing BigTable and TensorFlow, explains why:

> *One of the things that's really happened in the last 5 or 6 years, that has caused machine learning to really take off, is that we now have enough computational power, and large enough and interesting real-world datasets, to solve problems that previously we weren't able to solve in any other way: problems in computer vision, speech recognition and language understanding.*
>
> - Jeffrey Dean[4]

2.3.1 Supervised versus unsupervised learning

Supervised and unsupervised learning are two approaches to machine learning (Gollapudi, 2016) [12]. With *supervised learning*, the samples in the dataset are labeled. This means that for all inputs the correct output is known. For example, in an application where we want to recognize handwritten digits from images, we label all sample images with the correct answer, i.e. what digit is in what image. Supervised learning problems can again be grouped in *regression* problems, where the output is a real value, and *classification* problems, where the output variable is a category.

With the other approach, *unsupervised learning,* the examples are not labeled. In other words, there is no information on the 'correct' output. Considering the same example as before with the handwritten digits, the algorithm would not be able to answer exactly what

[3] https://research.google.com/pubs/jeff.html
[4] Interview with Jeffrey Dean on the Google Cloud Platform blog, February 1, 2017: https://cloud.google.com/blog/big-data/2017/02/jeff-dean-on-machine-learning-part-1-surveying-the-landscape

digit is in what image. However, it could try to *cluster* the data into different groups. Clustering is the most used form of unsupervised learning.

2.3.2 Training, validation and test dataset

As mentioned above, to perform supervised learning we need a labeled dataset. This dataset is generally split into three subsets: the training, validation and test set (Priddy & Keller, 2005) [14]. The *training set* is the set of samples and their correct answers that is shown to the model, which are used to find the optimal model parameters. The *validation set* is used to assess the performance of different models and/or approaches and to select the best performing approach. The *test set* is used to estimate the performance of the selected, final approach.

2.4 Deep Learning

Deep Learning is a subfield of machine learning that uses algorithms inspired by the structure of the human brain called Artificial Neural Networks (ANN), or often just neural networks. As with machine learning in general, the idea of ANNs is not recent. As early as 1957, Rosenblatt invented the perceptron algorithm, that he used for image recognition [4]. However, it would not be until the 2010s before data and computing power enabled neural networks to improve the state-of-the-art in many applications and become as popular as they are today. A big advantage of Deep Learning is that as we construct larger neural networks and train them with more and more data, their performance keeps increasing. In contrast, the performance of other machine learning techniques will generally reach a ‘plateau’ in their performance (Ng, 2015) [16]. See figure 1 for a depiction of this difference between Deep Learning and other machine learning techniques.

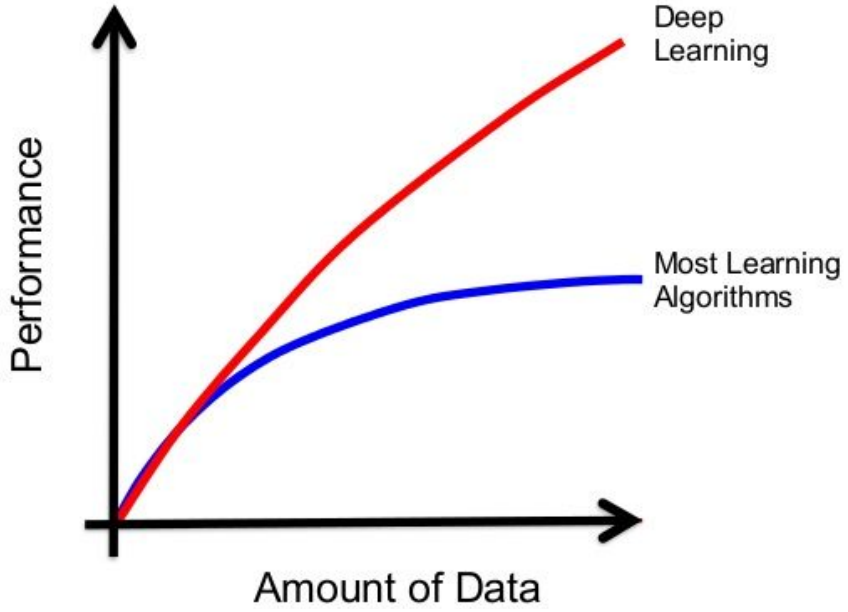

Figure 1: Performance of Deep Learning versus older machine learning algorithms

Another large advantage of Deep Learning over traditional machine learning is the ability to avoid something called feature engineering. One of the downsides of traditional machine learning is the need for good *features*. Features are input variables for the machine learning model. The success of a machine learning solution can heavily depend on finding good features. For example, suppose we want to use machine learning to determine whether a picture contains a land animal or a sea animal. We could select as a feature the ratio of blue pixels versus green pixels, knowing that many pictures of sea animals will contain many blue pixels and pictures of land animals will have relatively more green pixels. This process of selecting good features, often using knowledge of the domain, is called *feature engineering*. In practice, feature engineering is often difficult and expensive in terms of time and expertise (Längkvist, Karlsson, & Loutfi, 2014) [17].

Deep Learning is one of the only methods where the challenges of feature engineering can be avoided. This is because deep learning models are capable of learning to focus on the right features by themselves. For the example used above, we could simply feed the neural network many, many pictures of sea and land animals and it will figure out the relevant features by itself.

2.4.1 Neural networks

An Artificial Neural Network (ANN) is built up of *neurons*. A schematical representation of such a neuron is displayed in figure 2. Each neuron has a set of inputs x, and each of these inputs is given a specific weight w. Simply put, the neuron first calculates a weighted sum of these inputs:

$$z = x_1w_1 + x_2w_2 + \ldots + x_nw_n + b$$

Then, this z is put through f, the *activation function*. This activation function is a non-linear function, introducing non-linearity into the network and enabling the network to model non-linear dependencies between the target variable and the input variable(s). Note that a linear activation function *can* be used, but if *only* linear functions are used, the neural network would just behave like a linear regression, no matter how deep the actual network is. This is because the sum of any N amount of linear layers is just a linear function, and can therefore be replaced by a single layer where the activation function is a linear combination of the N activation functions we had before.

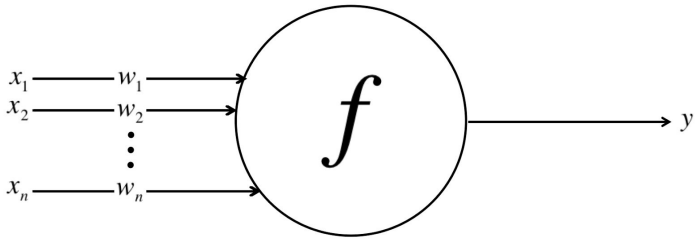

Figure 2: a neuron in a neural network

A neural network is created by connecting many of these neurons together. Neurons are often organized in layers and a neural network has at least three layers: an input layer, one or more hidden layers, and an output layer. An example of a neural network is depicted in figure 3. Note that there are an endless number of neural network configurations networks possible by adjusting the amount of neurons in each layer, adjusting the connectivity of the

neurons (note that not every neuron in one layer has to be connected to all neurons in the next as in the figure) and adjusting the amount of hidden layers.

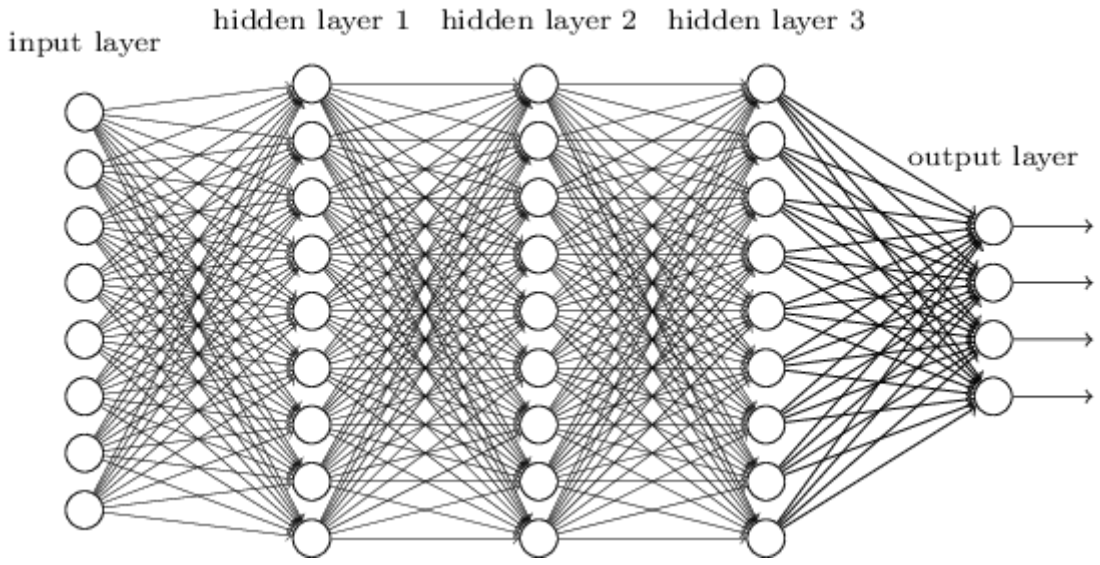

Figure 3: example of a neural network[5]

2.4.2 Training

Above, we have seen how a neural network can compute outputs using inputs, weights and activation functions. The hardest part of making a neural network is finding out what the values of those weights should be, which is done through training. In this paragraph an overview of this training process will be given.

At the start of the training process, the neural network will assign (random) starting values to all weights. Then, the neural network is shown the samples in the training dataset, consisting of both inputs and correct outputs. Using the input, the neural network computes an output and compares this to the correct output by calculating the error. This error function depends on the application, e.g. for regression problems the mean squared error is often used for the error function. Based on this error, the weights are adjusted so that if we would use the same input, the network's calculated output will be closer to the correct output. This process is done iteratively for all samples in the training set. Then, this procedure is done on the training

[5] Picture courtesy of Vahe Tshitoyan, retrieved from: https://nl.mathworks.com/matlabcentral/fileexchange/64247-simple-neural-network

set many times, hopefully finally converging on optimum values for the weights. The algorithm used to find these optimum values is called the optimization algorithm.

2.4.2.1 Optimization algorithms

The *optimization algorithm* is the algorithm that is used to update the neural network's weights to find their optimum values. More specifically, it is the algorithm that is used to minimize the error function using its gradient values with respect to the weights. There are many optimization algorithms, each with their own benefits and downsides. Most of these algorithms are variants of or extensions on *stochastic gradient descent.*

In short, standard *gradient descent* is an algorithm for finding the minimum of a function. In machine learning, this is often used because part of the training process is to minimize the error function, given a set of training data. This is done by iteratively updating the set of parameters until the minimum is found. With gradient descent, the parameters are updated after each pass over the entire dataset. Thus, when the dataset is very large, gradient descent might take a long time because each iteration the complete dataset is passed over. This is where *stochastic gradient descent (SGD)* improves over gradient descent (Bottou, 1991) [13]. With SGD, the parameters are updated as each sample is processed, and thus the model starts 'improving' from the first sample. This causes SGD to converge much faster than standard gradient descent.

2.4.2.2 Epochs

An *epoch* consists of one full training cycle over the entire training set. Optimization algorithms work iteratively; they need multiple passes over the training data set to converge to the optimum values for the parameters. Choosing too little epochs will result in a model that is underfitted, choosing too many can overfit the model.

2.5 Time series forecasting using Artificial Neural Networks

In the previous section we give an overview of the basics of neural networks. In this section we will look at how neural networks can be used for predicting time series.

2.5.1 Recurrent Neural Networks

Traditional neural networks assume that all observations are independent. However, in many applications this is not the case. In time series, generally an observation at time *t* will depend on the observations in previous time steps. This is where Recurrent Neural Networks (RNNs) can be useful, because RNNs can make use of sequential information.

RNNs are *recurrent*, because they have loops in them, allowing information to be passed from one step to the next. This can be seen in figure 4. In this figure, the neural net A takes x_t as input and outputs a value h_t. Then, information on the computation of x_t is passed on to the next step, where the neural net A will use this information on the computation of x_{t+1}.

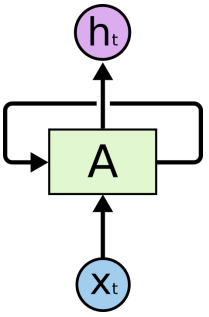

Figure 4: Recurrent Neural Network

In the time series example, this means that a computation for the observation at time *t* will be dependent on the computations of *t-1, t-2,* etc. Another way to look at this is that RNNs have a 'memory' in which it stores information about previous elements of the sequence.

As discussed above, the appeal of RNNs is the idea that they can use previous information in the current computation. However, the problem with regular RNNs in practice is that the farther in the past this relevant previous information lies, the harder it is for the RNN to learn this dependency. In other words, regular RNNs have trouble learning *long-term dependencies. Long short-term memory networks*[6] (LSTMs) are a type of RNN, specifically designed to solve this problem of regular RNNs (Hochreiter & Schmidhuber, 1997) [5].

[6] For an explanation of LSTMs, refer to
https://colah.github.io/posts/2015-08-Understanding-LSTMs/

2.6 Distributed Deep Learning

Training deep networks is a time-consuming process, sometimes requiring multiple days or even longer to train, depending on the network structure. Currently, most deep neural networks are trained using GPUs, mainly because they are very efficient for matrix multiplication, which is an important aspect of the training process. However, in practice the use of GPUs might not always be desirable. Firstly, GPUs are expensive, so the rent or purchase of GPUs might economically not be desirable if CPUs are already in place. Secondly, GPUs can only hold a relatively small amount of data in memory and data transfer from CPU to GPU is slow, so the use of GPUs might not give the desired performance upgrade, depending on the application. Lastly, the use of GPUs might not fit within the existing infrastructure or workflow, for example if programmers don't have experience with GPUs or prefer to use the resources that are already in place.

The latter is the case with the Atmospheric Composition Services platform, where it is preferred to leverage Apache Spark, which is already used extensively, for training the neural networks.

2.6.1 Model and Data parallelism

Model and data parallelism are two approaches to parallelizing algorithms that are often used in the context of learning model parameters using multiple processing units, i.e. cores or nodes.

With the data parallel approach, the model is replicated over multiple processing units, while the dataset that is used to train the model is split over these units. Each processing unit then uses its part of the data to train the model. After all units have finished training the same model using their part of the data, all the parameters are averaged to get the final model. Models that are trained on a very large amount of data benefit the most from data parallelism.

Model parallelism works exactly the other way around. The entire dataset is replicated over all processing units, while the model parameters are split up over the units. Each processing unit uses the entire dataset to train its subset of the model parameters. When all

units have finished, the different models are put together to get the final model. Models with a large number of parameters benefit the most from model parallelism.

2.6.2 DistBelief and Downpour SGD

In 2012, scientists at Google Brain presented DistBelief, a distributed framework that was able to train deep neural networks using a large-scale cluster of machines (Dean et al., 2012) [6]. Making use of both model and data parallelism, they stated that using this framework they could both greatly accelerate the training of models and train models that were larger than could be contemplated otherwise. One of the main contributions of their paper was the new method they call *Downpour Stochastic Gradient Descent (Downpour SGD)*, an asynchronous variant of standard stochastic gradient descent.

Stochastic gradient descent and its many variants and extensions are probably the most commonly used optimization procedures for training neural networks. However, although SGD improves over standard gradient descent, the sequential nature of the algorithm still makes it impractical in a distributed setting. This is why Dean et al. introduced *Downpour SGD*, an asynchronous variant of SGD that enables data parallelism. With Downpour SGD, multiple replicas of the same model are trained in parallel on different subsets of the training dataset. The model replicas communicate updates through one parameter server, which keeps the current state of all model parameters. Each model replica operates in parallel and publishes model weight updates to and receives updated parameter weights from the parameter server. The algorithm is *asynchronous* because the parameter server does not wait until it has received updates from all the workers, but updates the state of the model parameters for every update it receives. Dean et al. demonstrated successful training of large models with a new world-record accuracy on a visual object recognition task.

Since DistBelief, several other systems have been build that distribute the training process of neural networks. For example, Microsoft researchers used asynchronous SGD to build Project Adam, their own scalable deep learning training system [7]. In 2016, the researchers that made DistBelief built TensorFlow based on their experience with DistBelief [8].

2.6.3 Distributed Deep Learning with Apache Spark

In a real world data science project, training a neural network is often one of many steps. Before this step, the data must often be obtained and preprocessed. For the systems discussed above, due to their nature as custom systems, these steps must be done separately, possibly in a different framework. If the training step could be done using a general purpose framework such as Apache Spark, both the preprocessing and training could be done within the same framework. This could increase performances if this allows elimination of additional processing steps required to move from one framework to another. Furthermore, when such a framework is already in place, it may be easier for developers to implement the neural net training within the framework they already know rather than another they may be unfamiliar with. This is why frameworks for training deep networks with Spark have started to see development.

Since its creation, Apache Spark has been the big data framework of choice for machine learning. MapReduce, where Spark sought to improve over, involves many reading and writing operations to disk. This is needed because MapReduce has no awareness of the total pipeline of map and reduce steps, so does not know what data it could cache in memory. Instead, it flushes intermediate data to disk between each step. This overhead makes algorithms requiring many fast steps, such as iterative algorithms, unacceptably slow. Many machine learning algorithms are iterative algorithms, and thus not suited for MapReduce. Spark improved on MapReduce by keeping data in memory, improving the performance of iterative algorithms that access the same dataset repeatedly.

There are six projects implementing distributed deep learning on Spark. These projects can be found in table 1 below. Three of these projects are written in Python, two in Scala and one in both Scala and Java.

Library	**Year (date of first commit)**	**Language**
dist-keras	July 2016	Python
elephas	Aug 2015	Python

SparkNet	Nov 2015	Scala
CaffeOnSpark (yahoo)	Feb 2016	Scala (python api)
TensorFlowOnSpark (yahoo)	Jan 2017	Python
Deeplearning4j	Nov 2013	Java and Scala

Table 1: distributed deep learning frameworks on Apache Spark

Chapter 3

3 Data

The primary data source for this thesis is the Dutch OMI NO2 (DOMINO) data product from KNMI's Tropospheric Emission Monitoring Internet Service (TEMIS)[7]. This data is retrieved by the Ozone Monitoring Instrument. In this chapter the characteristics of the data used in this research will be discussed. In the first section, the characteristics of NO2 will be discussed and explained why NO2 is the pollution gas of choice for this research. The next section discusses the retrieval of the remote sensing data, from the satellite collecting raw observations to a quantified atmospheric composition. In the following section we will elaborate more on the uncertainty of remote sensing data, a characteristic that was briefly mentioned in the introduction. The final section of this chapter will discuss data flagging, the marking of 'bad' observations.

3.1 NO2

The OMI instrument measures many trace gases, but this research will use the OMI measurements for nitrogen dioxide (NO2) as the main data source. NO2 is a key component of air pollution that is emitted from any combustion process, such as power plants and vehicles that run on coal or gas. The only natural sources of NO2 are forest fires and lighting. However, more than 50% of the NO2 in the atmosphere is estimated to come from fossil fuel combustion. For this reason, and the fact that NO2 has a short lifetime in the atmosphere and thus does not get transported far from the emission source, NO2 measurements give a good indication of the location and amount of pollution caused by human activity (Beirle et al., 2003) [19].

[7] http://www.temis.nl/airpollution/no2.html

The unit that is used for the NO2 data is the *tropospheric vertical column density*. This density denotes the amount of NO2 particles in the troposphere at a certain location. The troposphere is the lowest layer of the atmosphere, it starts at sea level and has a height of 7-20 km, depending on location and season[8]. The amount of particles in the vertical column of an OMI pixel is in the order of magnitude of 10^{15}.

An important characteristic to note about NO2 is that there is a seasonal variation in NO2 in the atmosphere. In winter, NO2 concentrations are much higher in places with a high amount of pollution compared to the summer. There are two main reasons for this seasonal variation. The first reason is that in winter the combustion power plants are used more heavily to heat all homes in the colder winter period. The second reason is that NO2 stays in the atmosphere for a longer amount of time in winter compared to summer. This is due to the fact that chemical reactions of NO2 in the atmosphere are initiated primarily by sunlight. With less sunlight on average in winter compared to the summer, these reactions are less likely to take place and the NO2 stays in the atmosphere for a longer amount of time before it reacts.

3.2 Data retrieval

3.2.1 Swaths

Satellites collect remote sensing data in *swaths*. As a satellite revolves around the Earth, the sensor "sees" a certain portion of the earth's surface, as illustrated in figure 5. The area imaged on the surface, is referred to as the swath. The satellite's orbit and the rotation of the earth together allow complete coverage of the earth's surface, after it has completed one complete cycle of orbits. With OMI, the satellite orbits the earth 14 times per day, providing daily coverage of the entire earth.

[8] https://scied.ucar.edu/shortcontent/troposphere-overview

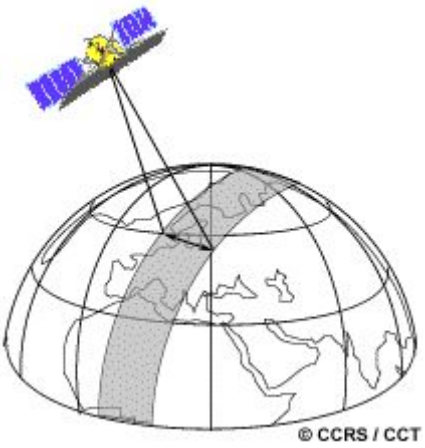

Figure 5: a swath

For illustration, the data collected in one swath as projected on a flat image of the earth can be seen in figure 6. In paragraph 3.4.1 below, the two stripes of missing data in the swath will be discussed.

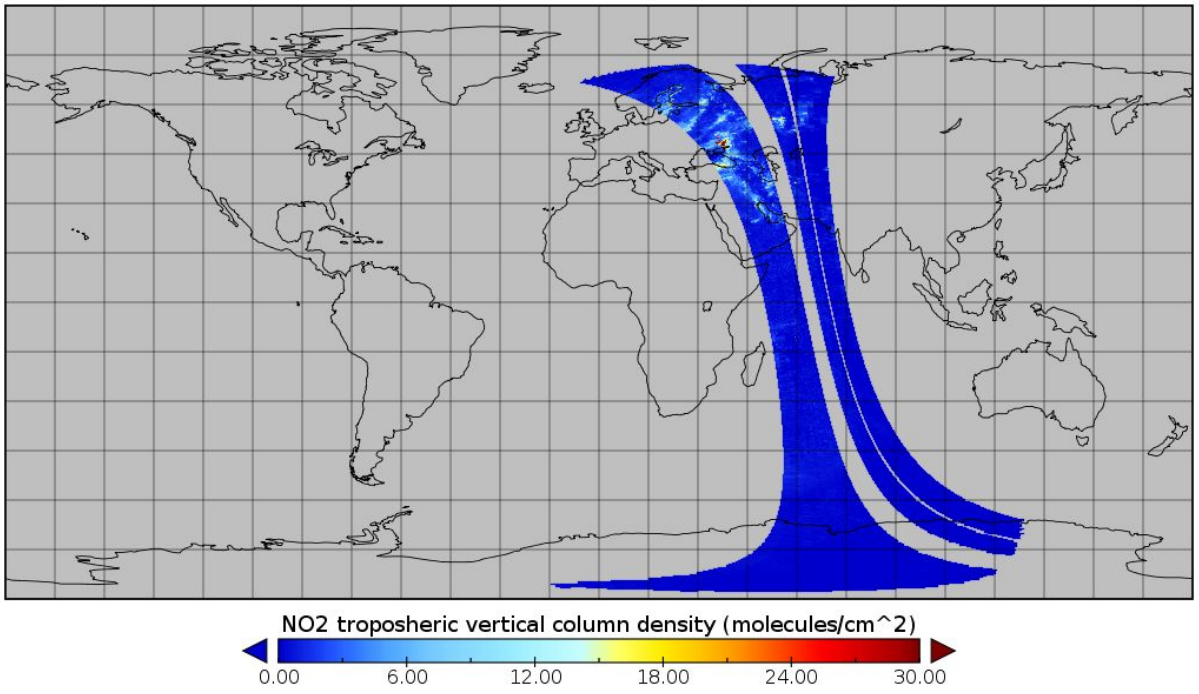

Figure 6: A swath projection

The OMI sensor's resolution is approximately 13 x 24km around the equator. Because of the curvature of the earth, these pixels do not all observe an identical, regular area on the earth's surface. To the edges of the swath, the pixels are more stretched out compared to the pixels in

the center of the swath. This can be easily seen in figure 7, which contains the same data as figure 6, but zoomed in to see the stretched pixels.

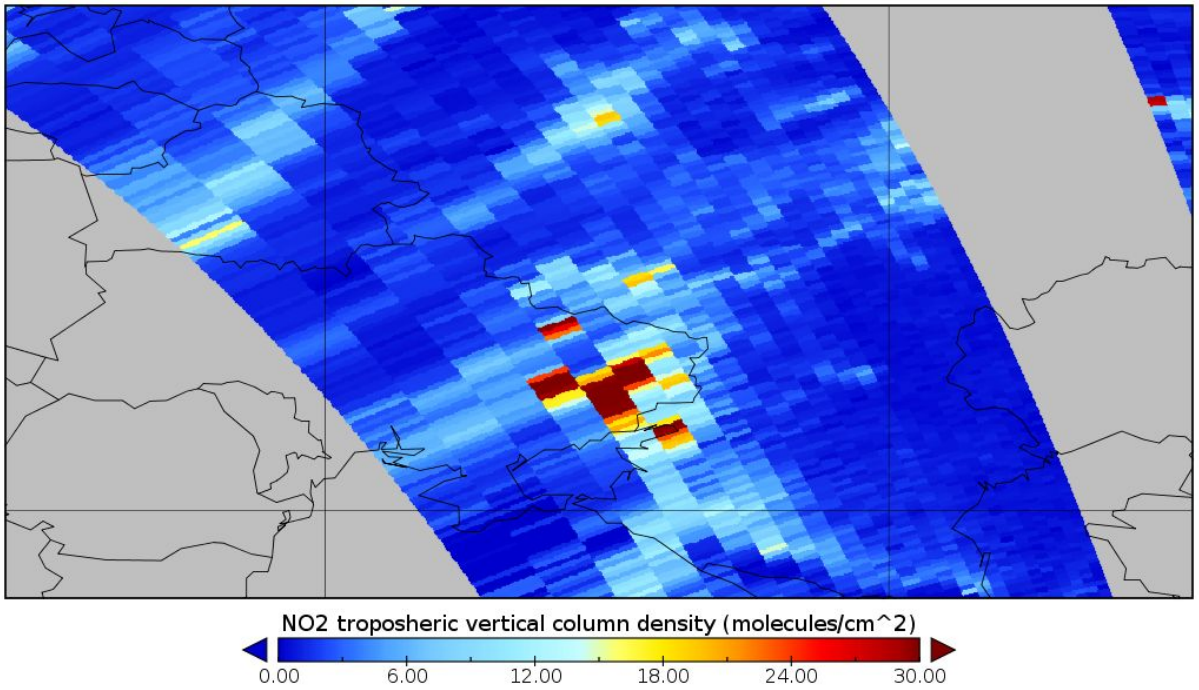

Figure 7: the OMI pixels are irregular

3.2.2 Product levels

The OMI data products are categorized in different levels. In table 2 below, the different levels are layed out. In this research Level 2 data is used.

Level	**Description**
Level 0	Raw instrument data
Level 1A	Raw instrument data with ancillary information (radio- and geometric calibration and georeferencing parameters) appended but not applied
Level 1B	Raw instrument data with ancillary information applied
Level 2	Atmospheric products derived from the raw data

Level 3	Gridded and quality controlled Level 2 data
Level 4	Output from models or results from lower level data analysis

Table 2: OMI product levels

3.3 Uncertainty

In the step from Level 1B to Level 2 data, to retrieve the vertical column densities for atmospheric products from the instrument's raw measurements, a method called DOAS[9] is applied. In this research we will not going into the technical details of this algorithm, but the important part to note is that this method applies two steps that both introduce uncertainty in the final vertical column density.

In the first step, a model is applied that derives a *slant column density* from the raw measurement. The slant column density is the amount of the trace gas along the path taken by photons from the sun, through the atmosphere to the earth's surface, and back through the atmosphere to the satellite's sensor.

In the second step, the slant column density is translated into a vertical column density using an *air mass factor*. This air mass factor is computed using parameters such as cloud fraction, cloud height and surface albedo as input. Due to errors in this input, e.g. an error in the cloud description, the retrieved air mass factor will have an error and thus the final result will also have an uncertainty. This step is the most important source of error in the final vertical column density.

Boersma et al. found that for significant tropospheric slant column density, a meaningful estimate of the tropospheric vertical column can be given with a precision of 35–60% [9]. These retrieval uncertainties are dominated by the uncertainty in the estimate of the tropospheric air mass factor. The most important uncertainties associated with the computation of the tropospheric air mass factor are cloud fraction, surface albedo and profile shape.

[9] Differential Optical Absorption Spectroscopy, https://sentinel.esa.int/web/sentinel/technical-guides/sentinel-5p/level-2/doas-method

3.4 Data flagging

Remote sensing is impacted by many variables that influence the quality of the data retrieval. In this paragraph a few of these variables and their impact on the dataset are discussed.

3.4.1 Row anomalies

Since June 2007, the data from OMI is affected by so-called row anomalies. The origin of the row anomalies is unclear at the moment, but there are indications that the instrument's field of view is partly obstructed. Various row anomaly correction algorithms have been developed since the first occurrence, but to date, no satisfying correction has been implemented that removes the anomalies. Therefore, the affected rows are discarded as they are not fit for scientific use, based on the flag that is raised for these rows in the OMI data product[10]. In figure 6, the row anomalies can be easily seen by the two strips of missing data.

3.4.2 Cloud fraction and surface albedo

For the retrieval of tropospheric trace gas columns, information on the cloud cover conditions is also needed. Both the cloud fraction and the cloud pressure are used as input for the algorithms that calculate the trace gas concentrations from the raw measurements of the OMI instrument. In the DOMINO data product, a flag is raised if the cloud cover is larger than 50%.

Surface albedo is the fraction of the sunlight reflected by the surface of the earth. For high surface albedo, the cloud retrieval is unreliable. Following KNMI's recommendation, all observations with a surface albedo larger than 30% are discarded.

[10] As prescribed in the data product description: http://www.temis.nl/docs/OMI_NO2_HE5_2.0_2011.pdf

Chapter 4

4 Implementation

In the context of this research, two use cases for the Atmospheric Composition Services platform have been implemented. In this chapter, these two implemented use cases will be discussed, starting with regridding in the first section and then air quality forecasting in the next. Preliminary experiments have been performed for two other use cases, which will be discussed in the appendix.

4.1 Regridding

With this use case, the objective is to *regrid* the irregular satellite data, consisting of observations in polygon-shaped pixels, to a regular grid. Such a regular grid should be produced for each day in the entire OMI dataset, with a target processing time of approximately one minute per day. The grid size for this use case is 0.125 by 0.125 degrees.

4.1.1 Method

4.1.1.1 Filtering

Before starting the regridding algorithm, the data is filtered based on the parameters discussed in 3.4. All observations that have a flag raised in the OMI data product due to the row anomalies or cloud fraction are discarded, as well as all observations with a surface albedo larger than 30%.

4.1.1.2 Algorithm

To be able to fully parallelize the regridding of the satellite data using Apache Spark, the algorithm is designed to be entirely independent **for each satellite observation**. In figure 8

below, one satellite observation is depicted with the light-blue polygon. The black, squared grid is the regular grid we want to grid the satellite data to.

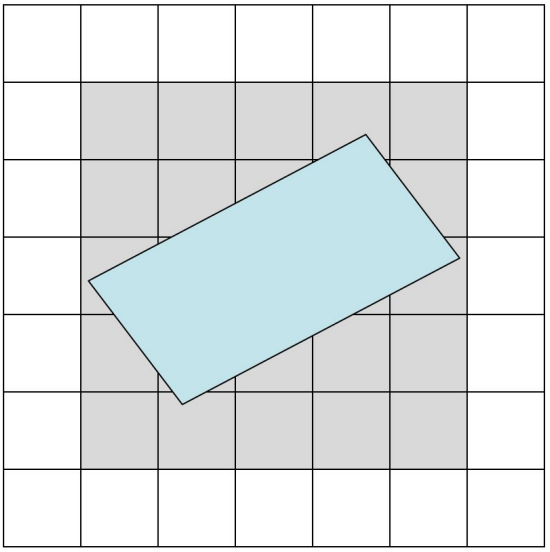

Figure 8: schematic depiction of OMI pixel (blue) on regular grid (black)

The algorithm for the regridding is as follows:

1. Get the four cornerpoints of the satellite pixel.
2. From these cornerpoints, compute all gridcells the observation *can possibly intersect*, depicted in figure 8 with the grey area.
3. For all gridcells in the grey area:
    3.1. Compute the area of overlap between the gridcell and the satellite pixel
    3.2. Add the value of the observation to the gridcell, with a weight equal to the overlapping area
4. Return all gridcells that overlap with the pixel

This algorithm is applied to all satellite observations for one day, after which a weighted average is taken for each gridcell. These averaged gridcells are then combined to end up with one grid per day.

4.1.1.3 Cube data structure

The daily grids that are the result of the regridding algorithm, are saved in a data structure we call a **Cube.** This data structure is implemented using the netCDF libraries[11]. The Cube stores the layers in a netCDF file with three dimensions: latitude, longitude and time (in days). A Cube API is implemented for the querying of data from the structure, so that future users can easily implement their own applications using the Cube data structure. On top of reading and writing entire daily grids to and from the Cube, methods are implemented that allow the query of timeseries for given areas and over given intervals, e.g. querying all data in the Netherlands in the year 2015. Optionally, the returned time series is averaged over a certain period (e.g. return the weekly averages instead of daily values) or area (e.g. return a time series averaged over the Netherlands instead of separate grid cells).

4.1.2 Results

The speedup in the amount of cores is used to assess the scalability of the regridding implementation. The average processing time for one day using 1, 2, 4, 8, 16, 24 and 32 cores is measured, separately for days where the row anomalies were not present yet (before 2007), and days where they were present. Each machine has 8 cores, so for the processing useing 16, 24 and 32 cores the communication between machines will start to impact the performance. Then, the speedup is calculated by dividing these average processing times by the average processing time using one core.

In figure 9 below, the blue line represents the speedup for days where the row anomalies were not present yet, the red line the speedup where they were. The grey line represents the ideal speedup, if doubling the amount of cores would mean a halving of the processing time. We can see that the ideal speedup is achieved up until 4 cores, after which the Spark overhead starts to impact the speedup. With 32 cores, the speedup is around 16 times. With this speedup, the processing time per day is lower than the goal of 1 minute per day.

[11] https://www.unidata.ucar.edu/software/netcdf/

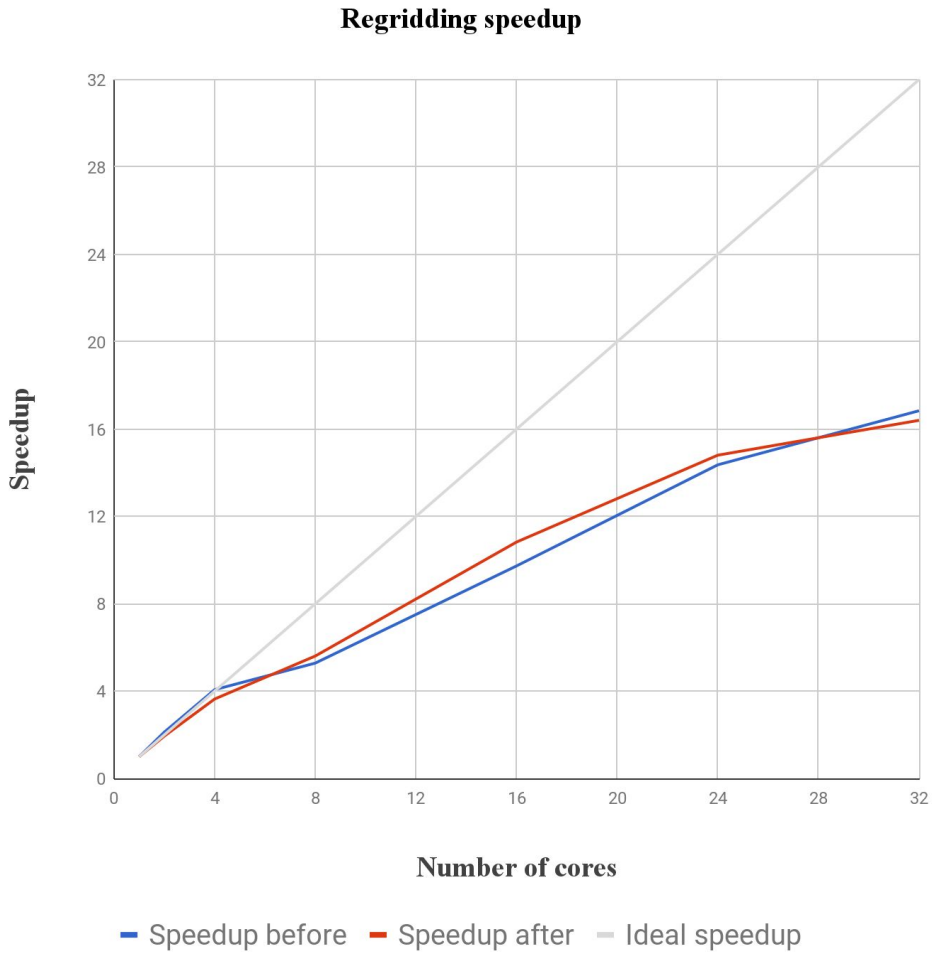

Figure 9: time spent processing one day of satellite data

To illustrate how the gridded data looks, figure 10 depicts the average NO2 vertical columns over 2010.

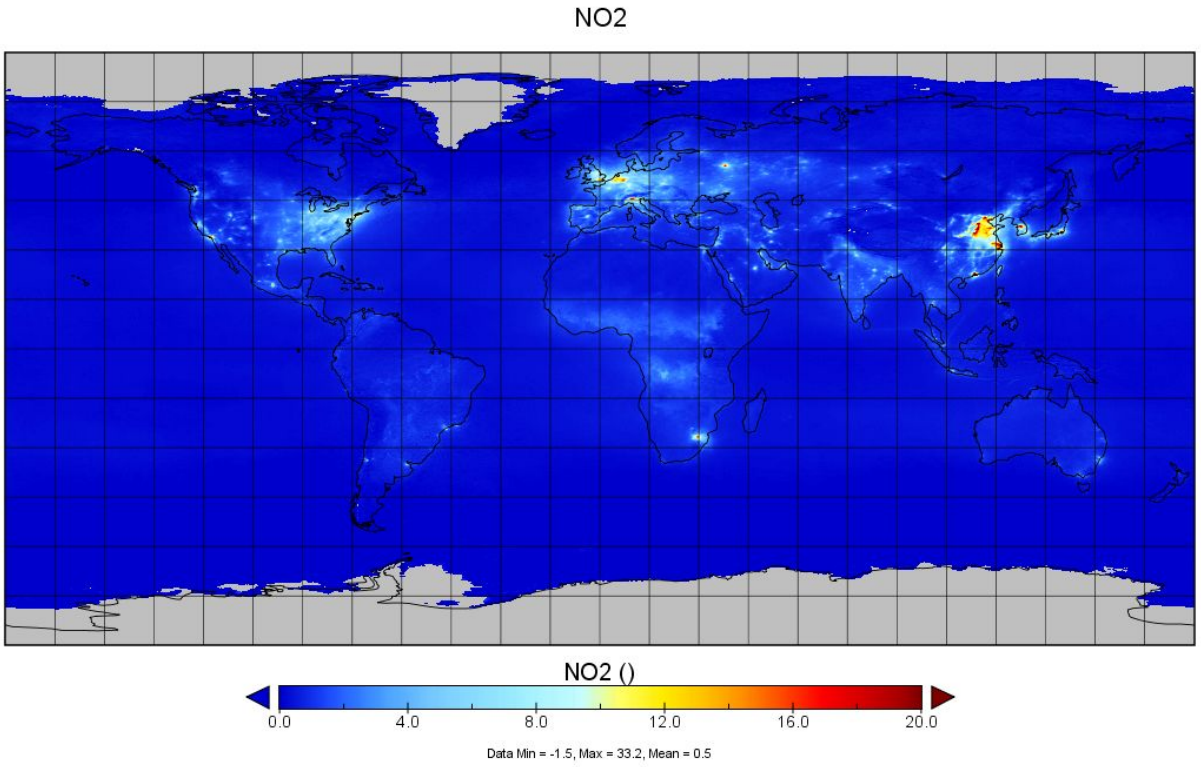

Figure 10: Average NO2 Vertical Column over 2010

4.2 Air quality forecasting

For this use case, the goal is to calculate expected trace gas concentrations in the future using machine learning. First, a base model is fit to the data which will be used to compare the performance of the deep learning models against. Then we will explore if we can use neural networks to achieve similar or better results compared to the base models. For all models NO2 data will be used from 2005 to 2016, and the dataset will be split in a training (50%), validation (25%) and test (25%) set. The data is split over time, so that the training set will contain the data for 2005 through 2010, the validation for 2011 through 2013 and finally the test set the data for 2014 through 2016. The training set is used to train the model, the validation set is used to tune the hyperparamaters of the model (e.g. neural network structure) and finally the test set is used to assess the performance of the final model.

In section 3.1, we discussed that the NO2 data has a seasonal trend. This can seasonal trend can be shown by doing a seasonal decomposition, using an additive model. The additive model is of the following form:

$$Y[t] = T[t] + S[t] + e[t]$$

Here Y is the observed value, T the trend, S the seasonal variation and *e* the residual. To illustrate, a seasonal decomposition is shown for a gridcell that contains the biggest coal factory in India in figure 11. For this decomposition the seasonal_decompose method was used in the Python statsmodels package.[12] Figure 11-A depicts the observed data, where missing data is filled by linear interpolation. Figure 11-B depicts the trend that denotes the long-term progression of the series, calculated by using moving averages. Figure 11-C shows the seasonal variation, and lastly Figure 11-D the residuals.

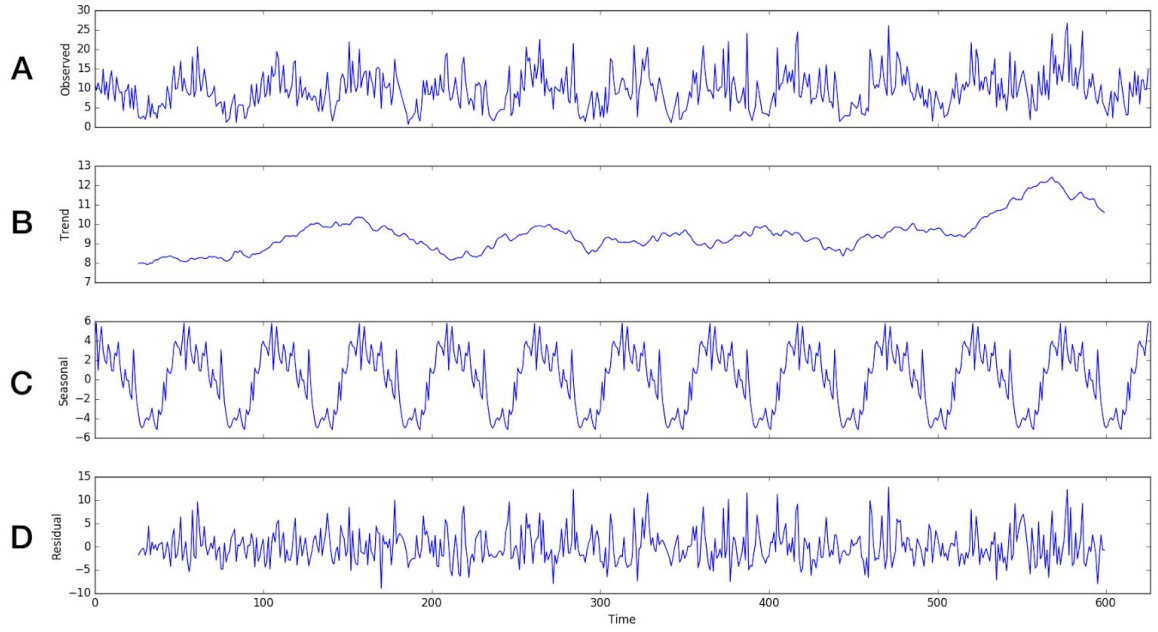

Figure 11: Seasonal decomposition for one gridcell in India

Model evaluation

For the evaluation of continuous variables, the Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE) are the two most common metrics. Both MAE and RMSE are inverted scores, i.e. the lower the metric the better the model.

The MAE measures the average magnitude of the errors in a set of predictions, without considering their direction. It's the average of the absolute differences between predicted values and actual observations where all individual differences have equal weight.

[12] http://www.statsmodels.org/

The RMSE on the other hand gives relatively more weight to large errors, because the errors are squared before they are averaged. Thus, the RMSE penalizes large errors more than small errors and the MAE penalizes all errors equally. To illustrate, first suppose we made two predictions, both with an error of 5.5. The MAE will just be the average of the two errors, so $MAE = \frac{5.5 + 5.5}{2} = 5.5$. The RMSE will be $RMSE = \sqrt{\frac{5.5^2 + 5.5^2}{2}} = 5.5$. Now suppose we make another two predictions, but now one has a smaller error of 1.0 and one a larger error of 10.0. Now, the MAE does not change: $MAE = \frac{1.0 + 10.0}{2} = 5.5$. However, the RMSE will be higher than before: $RMSE = \sqrt{\frac{1.0^2 + 10.0^2}{2}} = 7.11$. We see that the RMSE gives more weight to higher errors, where for the MAE it does not make a difference.

In this research the MAE is chosen as the primary metric to evaluate the models for two reasons. Firstly, the MAE is easier to interpret. Secondly, as stated above, the RMSE penalizes large errors more. As our data source is prone to have outliers because of relatively large uncertainty in the observations itself, large errors in the model have a high probability to come from an error in the observation rather than the model. With that in mind, penalizing these larger errors more than smaller errors does not make sense.

Although the MAE does not penalize large errors more than small errors, the large uncertainties in the NO2 observations are expected to make the MAE pessimistic in regards to the *actual* NO2 concentrations. This is however a constraint of the data retrieval and not of the model or model evaluation itself.

To compare the base model with the neural networks, three areas of interest have been chosen, because fitting the neural network for all gridcells globally is not feasible from a time perspective. Over these areas the average, relative MAE is taken for all three methods to decide which performs best. The chosen areas and their latitude and longitude ranges can be found in table 3 below.

Area	**Longitude range**	**Latitude range**
India's largest coal plant	[262.500, 263.750]	[24.375, 23.125]
Riyadh	[225.625, 226.875]	[25.000, 23.750]
Sahara desert	[177.500, 178.750]	[28.750, 27.500]

Table 3: areas of interest

4.2.1 Base model

4.2.1.1 Method

Based on the above seasonal decomposition, we come up with a simple base model that incorporates both the trend and seasonal variation. We can use a sine wave to estimate the seasonal variation, and the trend can be approximated by a linear component. Then, the base model that is fit to the data is of the following form:

$$\hat{y} = a + b \cdot t + c \cdot sin(2\pi tf + d)$$

In this model, the parameter *t* represents time and *f* represents the frequency of the sine wave, which is fixed to one cycle per year. The parameters *a, b, c* and *d* are the parameters that are estimated using the training data. The parameter *a* represents the vertical shift of the sine wave, which normally oscillates around 0 and will be shifted according to *a*. The parameter *b* represents the linear trend. For example, a value of b=0.1 would mean that the NO2 concentration increases with 0.1 every *t* time units. Finally, the parameters *c* and *d* represent the amplitude and the horizontal shift of the sine wave, respectively.

Using this model for forecasting is straightforward, as this method simply fits a curve to the training data. After that, the only input parameter that is needed is the parameter *t*, the number of weeks since the first week used in the training data.

A separate model is estimated for all gridcells, meaning that the time series for each separate gridcell will be used to estimate a model with its own parameter values . The parameters are estimated using the python package Scipy[13], and using the nonlinear least-squares estimation method. For this estimation, the training dataset is used. The performance of the models is then evaluated by making a prediction on the test dataset and calculating the average Mean Absolute Error for each of the areas of interest. As this process must be executed for all gridcells, Spark is used to distribute the model estimations to keep the processing time within reasonable limits.

[13] https://docs.scipy.org/doc/scipy/reference/index.html

4.2.1.2 Results

To get an idea of the fit of the base model on the data, the base model results for one gridcell in each of the areas of interest are depicted in figure 12, 13 and 14. In these figures, the blue lines depict the data as measured by the satellite and after regridding (see section 4.1). The green, red and lightblue lines display the model fit for the training, validation and test set respectively.

We can see that the sine wave to capture the seasonality of the data works reasonably well for India's coal plant and Riyadh. For Riyadh, where the relatively high peaks can not be modeled by a (symmetrical) sine wave, it looks to work a bit less well than for India.

However, for the gridcell in the Sahara desert the regular sine wave seems less appropriate, especially for the last half of the model. The reason for this is that the seasonal effect on the NO2 concentration is much lower here, because the two main causes for NO2 seasonality are negligible here. Firstly, colder winter months increasing emissions is probably not relevant here, because there is no significant human activity in the Sahara desert. Secondly, more sun hours in the summer causing the NO2 in the atmosphere to react and consequently the concentration to decrease is also not relevant, as the Sahara desert is sunny all year round.

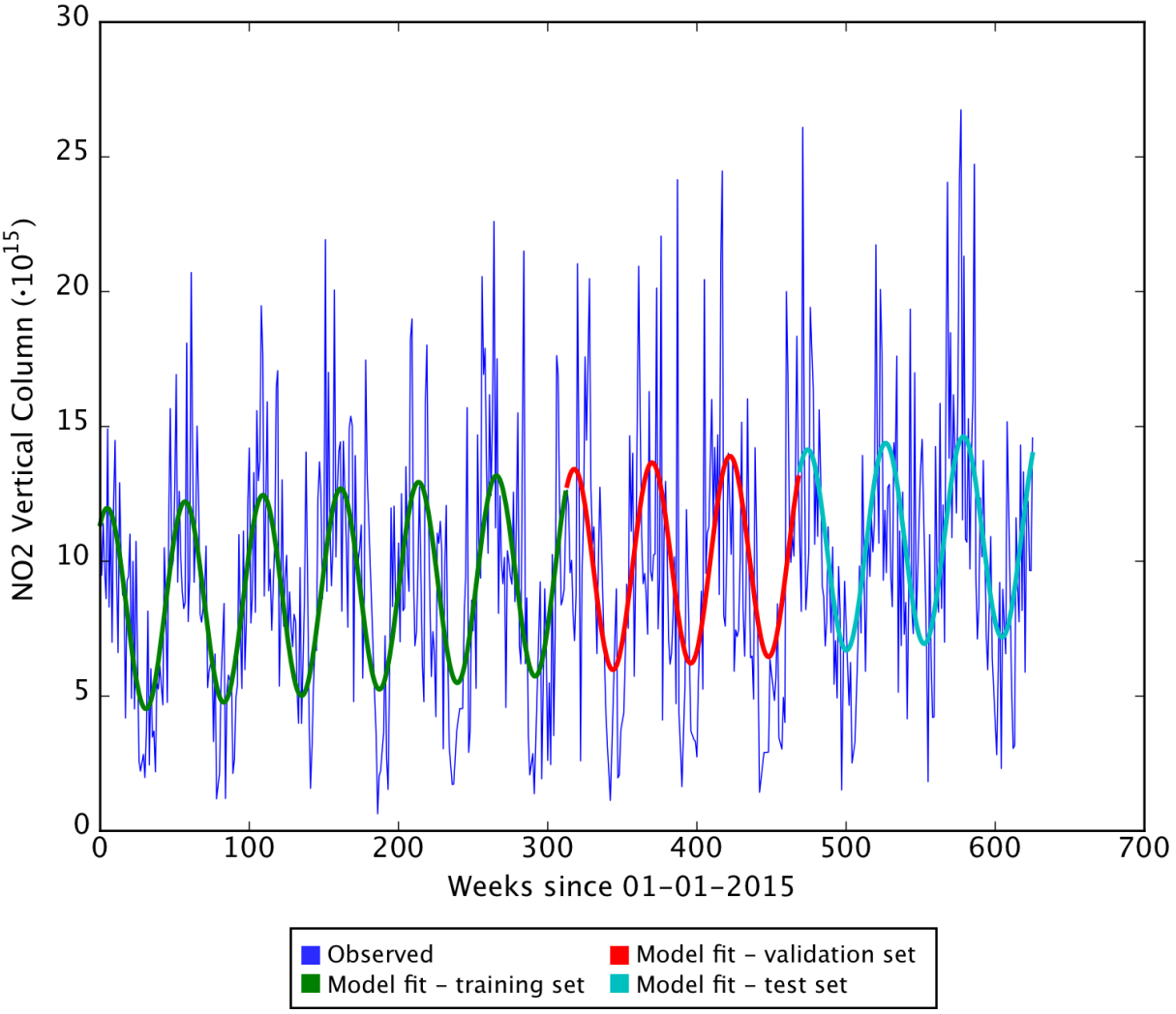

Figure 12: Base model result for India's largest coal plant

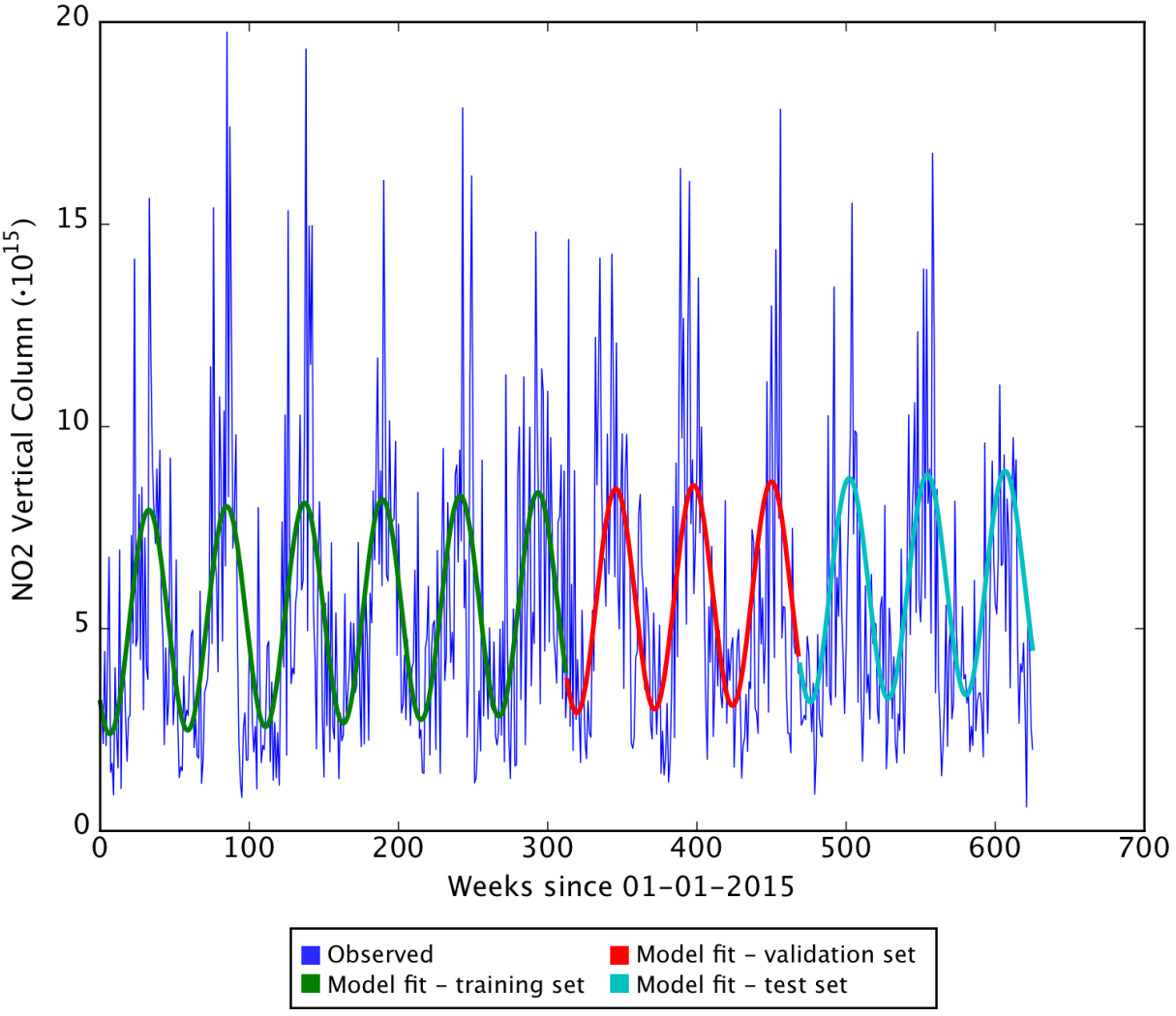

Figure 13: Base model result in Riyadh

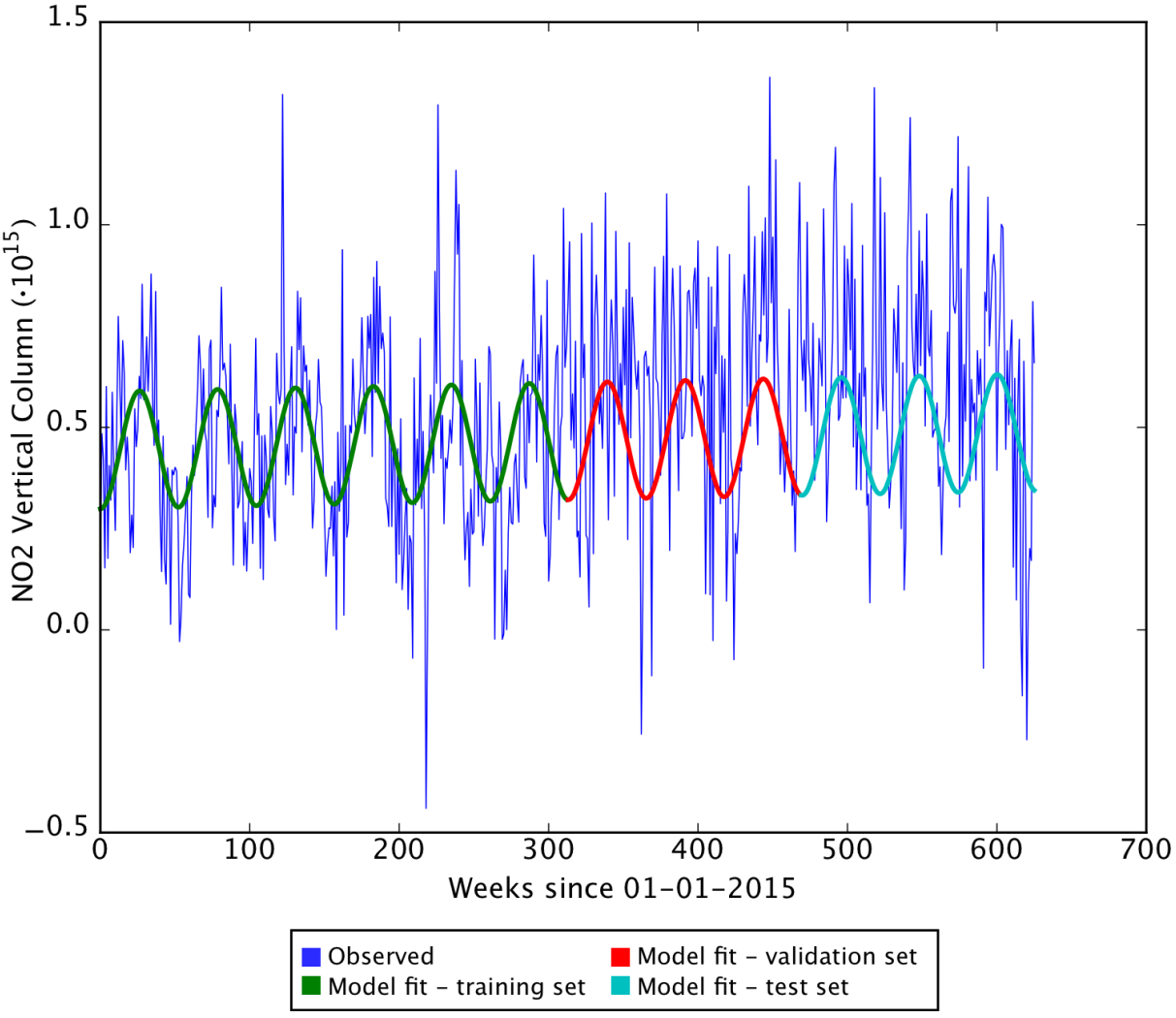

Figure 14: Base model result in the Sahara desert

Another way to illustrate this is by comparing the MAE for a simple linear fit (i.e. a straight line) through the data with the MAE of the base model. In table 4 below the average relative MAE is displayed for the three areas of interest.

Area	**Linear fit average MAE**	**Base model MAE**
India's largest coal plant	43.7%	39.3%
Riyadh	56.7%	47.2%
Sahara	38.5%	37.8%

Table 4: areas of interest's average relative MAE for base and linear model

We can see here that, for the areas of India's largest coal plant and around Riyadh, the base model improves over a simple linear fit. The average relative MAE for the base model is 4.4 and 9.5 percentage points lower, respectively. However, for the Sahara desert the average relative MAE only decreases with 0.7 percentage points. In this area the seasonal effect is negligible, and the variation is caused by meteorological effects and errors in the observations.

Another observation from the table is that the base model performs better for the coal plant in India than it does for Riyadh, with average relative MAE's of 39.3% and 47.2%, respectively. One explanation for this could be that the seasonal effect for the former is more clearly noticeable because the coal plant is directly responsible for generating extra energy if that is needed in winter months. Moreover, because the largest coal plant in India is responsible for powering a large area in India, the effect of the seasons is effectively averaged and therefore are smoother sine curve can be expected than measuring the seasonal effect in one location.

4.2.2 Seasonal ARIMA

As discussed in the background in section 2.2, the most used statistical method for time series forecasting is the ARIMA model. In this research, an extension of the ARIMA model, the seasonal ARIMA mode, is used, which is capable of dealing with time series that have a seasonal trend.

4.2.2.1 Method

Separate seasonal ARIMA models are fit over the areas of interest discussed above. The used model is an ARIMA(1, 1, 0)(1, 1, 0)$_{52}$ model. The meaning of these parameters can be found in table 5 below.

Parameter (bold and underlined)	**Description**
ARIMA(**1**, 1, 0)(1, 1, 0)$_{52}$	First order non-seasonal autoregressive term. This means that

	for time *t* the observation at time *t-1* is used as a regression term.
ARIMA(1, **1**, 0)(1, 1, 0)$_{52}$	First order integration. This means that first order differencing is used to make the time series stationary. First order differencing means that as dependent variable not the observation at time *t* is used but the difference between observations *t* and *t-1*.
ARIMA(1, 1, **0**)(1, 1, 0)$_{52}$	Zero order moving average term. This means that no past error is taken as regression term.
ARIMA(1, 1, 0)(**1**, 1, 0)$_{52}$	First order seasonal regressive term. This means that for time *t* the observation at time *t-S* is used as a regression term, where *S* is the number of periods in the seasonal pattern (defined later).
ARIMA(1, 1, 0)(1, **1**, 0)$_{52}$	First order seasonal differencing, removes seasonal trend. First order differencing means that as dependent variable not the observation at time *t* is used but the difference between observations *t* and *t-S*.
ARIMA(1, 1, 0)(1, 1, **0**)$_{52}$	Zero order seasonal moving average term. This means that no past error is taken as regression term.
ARIMA(1, 1, 0)(1, 1, 0)$_{\mathbf{52}}$	*S = 52*, meaning that there are the seasonal pattern takes 52 time periods. Because weekly averages are used, *S = 52* means that the seasonal pattern repeats once every year.

Table 5: ARIMA parameters

4.2.2.2 Results

To illustrate the fit of the seasonal ARIMA models on the data, the model prediction for the center cell in each of the three areas of interest is depicted in figure 15, 16, and 17 below.

From the figures, we can see that the predictions by the seasonal ARIMA model have a seasonal trend and the shape of each seasonal cycle is identical to the other cycles, similar to the base model. However, in contrast to the base model, the ARIMA model is able to model more irregular patterns than the base model, which only uses a regular sine wave to predict the seasonality. A possible benefit of this can be seen in the ARIMA model for Riyadh, in figure 16. With the base model, we found that the relatively high peaks could not be modeled by the (symmetrical) sine wave. The ARIMA model's capability to model these peaks seems to be better from the figure.

For the cell in the Sahara in figure 17, we see the same potential problem as before with the base model, where there is a low seasonal effect but the way the model is built, it still forces a seasonal trend. This may potentially have a negative effect on the performance of the ARIMA model in areas where there is negligible seasonality.

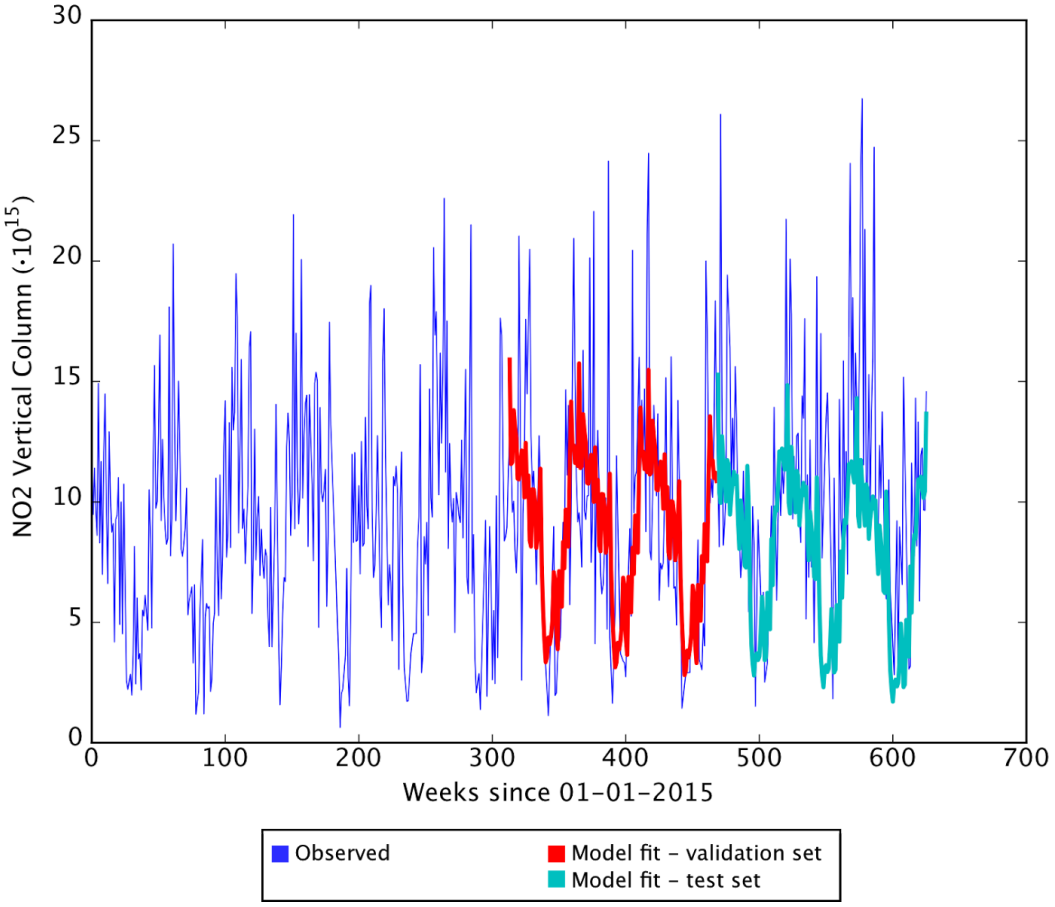

Figure 15: ARIMA model result at India's largest coal plant

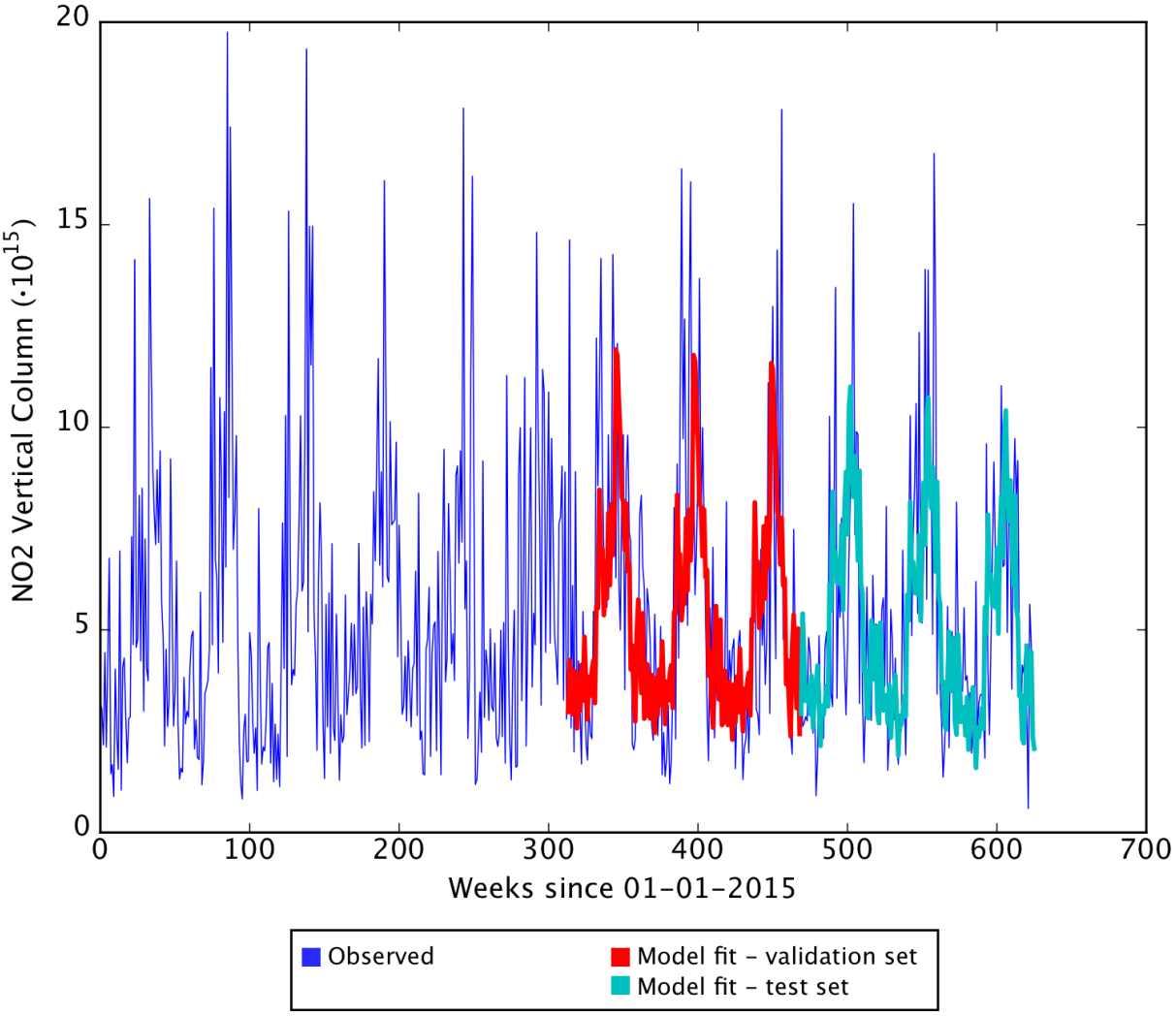

Figure 16: ARIMA model result in Riyadh

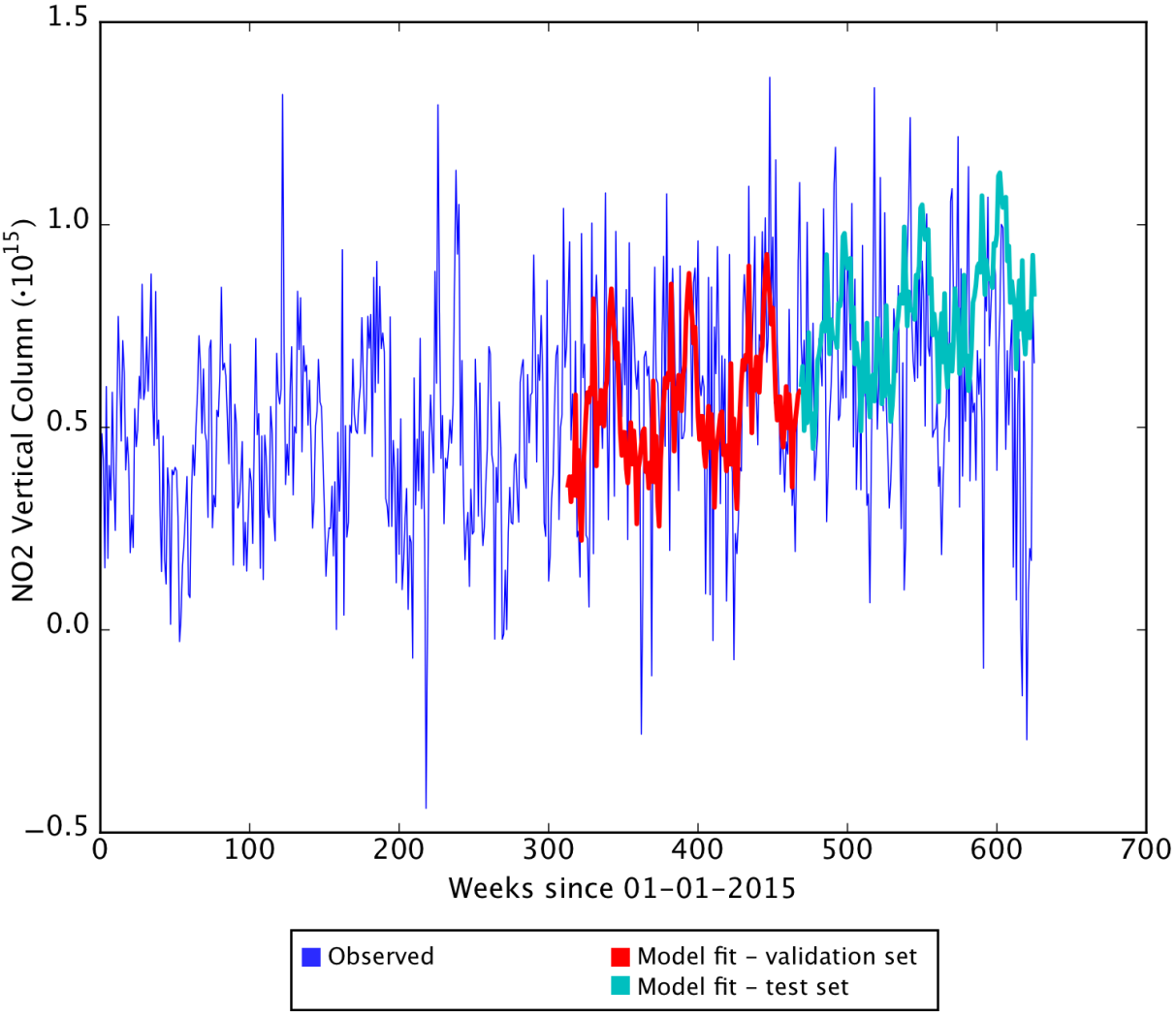

Figure 17: ARIMA model result in the Sahara desert

To quantify and compare the performance of the ARIMA model, the average relative MAE has been calculated over the three areas of interest. These values can be found in table 6 below, as well as the relative average MAE for the base model, for comparison.

It is noticeable that the ARIMA model significantly outperforms the base model for the area in Riyadh, improving over the base model's MAE with 3.5 percentage points. As discussed above, the cause of this improvement is expected to be the capability of the ARIMA model to model irregular seasonal patterns, as for example the relatively high peaks we saw in Riyadh. At India's largest coal plant, this seasonal effect is in itself more similar to a sine wave[14] and thus the base model performs reasonably well there. The ARIMA model is not able to improve over the base model's performance in this area.

[14] As explained in section 4.2.1.2

Finally, in the Sahara desert, the base model performs better than the ARIMA model. The ARIMA model's performance is even worse than a linear model in this area, and we saw before that the base model only slightly improved over a linear fit. This is to be expected however, as the variance in this area is expected to stem entirely from short-term meteorological effects and measuring errors. A seasonal trend is negligible here. Thus, models that force the prediction of a seasonal trend are not expected to outperform a linear model.

Area	**Linear fit MAE**	**Base model MAE**	**ARIMA MAE**
India's largest coal plant	43.7%	39.3%	41.2%
Riyadh	56.7%	47.2%	43.7%
Sahara	38.5%	37.8%	40.9%

Table 6: relative average MAE for base and ARIMA model

4.2.3 Neural network

In the base model above, the seasonal variation is estimated by a sine wave and the linear trend by a linear curve. However, in figure 11, we can see that the seasonal variation is not perfectly sinoid, nor is the trend linear. This raises the question if we can improve the model by using a neural network, that is able to learn nonlinear patterns. In this research, we use Keras[15], a neural network Python API, for the training of the neural network models. It runs on top of other neural network libraries, but with an API that is higher level and enables fast experimentation. In this research, we run Keras on top of TensorFlow, Google's neural network library. Apache Spark is used in combination with Keras to parallelize the training of a large amount of models.

4.2.3.1 Method

For neural networks, the structure and learning procedure of the network can have a significant impact on the performance of the final model. Unfortunately, few rules of thumb

[15] https://keras.io/

exist to decide what value to give these so-called hyperparameters. Therefore, in this research, many models with different hyperparameters will be trained and their performance will be tested to find the best hyperparameters for the problem at hand.

One important assumption is made in this research, which is that one model structure and training procedure is appropriate for all possible locations. In other words, the hyperparameters that we decide on by trying different configurations in a few locations, are usable for gridcells in all locations.

A decision needs to be made for the following hyperparameters: optimization algorithm, amount of hidden layers in the network, amount of neurons per hidden layer, amount of epochs.

Network structure

The most important decision when building a neural network model is the structure of the neural network itself. An endless amount of configurations are possible, and the chosen configuration significantly impacts the performance of the model. First we choose the type of neural network layers to be used. In our case we choose an LSTM network, because LSTMs are able to learn both short and longer term effects, as discussed in Chapter 2. Then, a decision needs to be made on the number of hidden layers and the number of neurons in each layer. For both these hyperparamaters a range was established by performing preliminary experiments, wherein the model performance was reasonable. Within these ranges, multiple configurations were tested to decide on the best performing model.

Optimization algorithm

The optimization algorithm can significantly influence the performance of the final model, based on the application. In this research, all optimization algorithms in Keras are compared for the final neural network structure to see their impact on the performance of the final model and the most appropriate algorithm is chosen.

Epochs

To choose the amount of epochs in this research, a model is fitted over a large amount of epochs, and after each epoch the performance of the model on the test set is evaluated. After

all epochs, the best performing model and corresponding amount of epochs is chosen from all evaluations.

Rolling forecast

The stateful LSTM used in this use case will be given one input and will return one output. In other words, if we feed the neural network input *t,* then it returns the prediction for *t+1*. Because we are interested in predicting not one but multiple timesteps ahead, we will use a *rolling forecast.* This means that if we want to predict *n* timesteps ahead, we will feed the neural network the input *t*, then use its prediction for *t+1* as input to get a prediction for *t+2,* etcetera.

4.2.3.2 Results

Hyperparameters

For the optimization algorithm, all available options in Keras have been tested. In figure 18 below, the MAE on the test set is depicted over the training epochs. We can see that a lot of the optimization algorithms result in a noisy curve. This is undesirable, because when the performance of the neural network varies greatly between two consecutive epochs, the performance of the final model will be somewhat arbitrary based on the chosen amount of epochs. For this reason we have picked the *adagrad* optimization algorithm for this research, as it is not noisy and the smooth curve reaches a minimum after some amount of epochs.

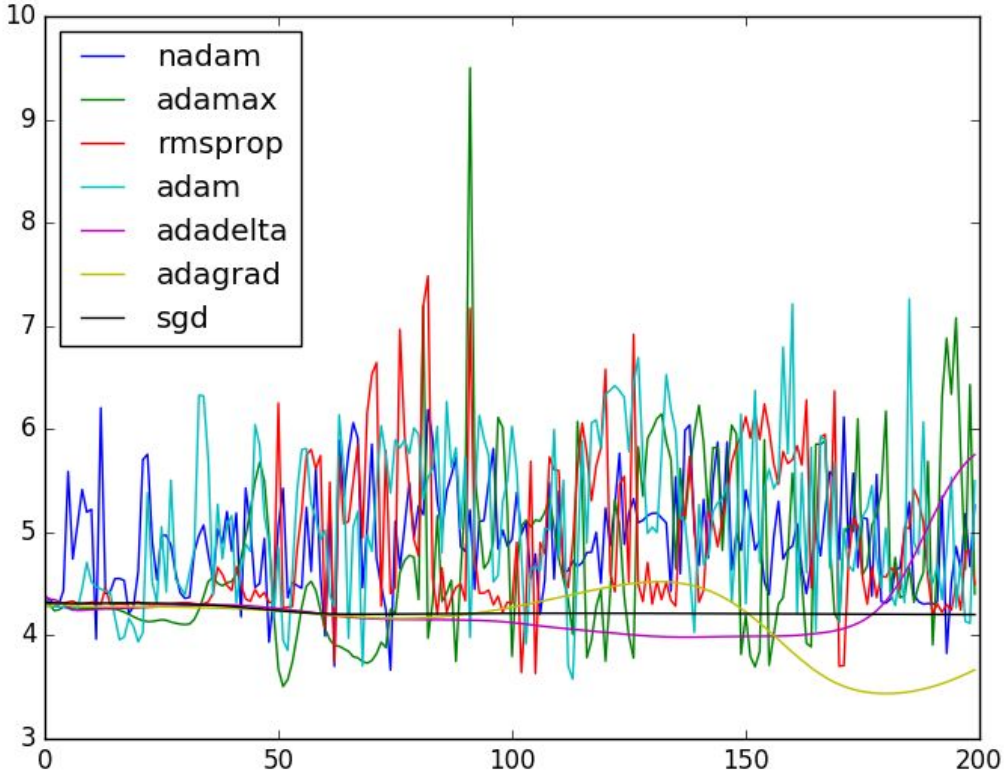

Figure 18: MAE over training epochs for all optimizers

For the network structure hyperparamaters, i.e. the amount of hidden layers and amount of neurons per hidden layer, a range was established by preliminary experiments within which the model performed best. Within these ranges, the model was trained using many different configurations. Then, the values for these hyperparamaters were decided upon based on the configuration that performed best on the validation set. These values are used for all gridcells, assuming that the same network structure is usable for all gridcells.

Finally, the optimal amount of training epochs was decided for every gridcell separately. During the training process, the performance on the validation dataset was evaluated after every epoch. An upper limit of 200 epochs was established as experiments proved that training even more epochs only decreased the performance on the validation set. After the completion of the training process, the amount of epochs was chosen that had the best performance on the validation set.

Model results

To get an idea of the fit of the neural network model on the data, the model results for one gridcell in each of the areas of interest are depicted in figure 19, 20 and 21.

In the first model, we can see the possible advantage of the neural network in that it is not bound to a regular sine wave to catch the seasonality in the data. This could potentially lead to a better performing model compared to the base model.

In the second model however a disadvantage of the neural network becomes clear. Although the model performs well on the validation set (green line), the model does not generalize well and has a poor performance on the test set.

In the third model, as well as the second, we can see that the neural network not necessarily resembles a sine wave. This may potentially lead to better performance in areas where the seasonality of NO2 is negligible, such as in the Sahara desert, compared to the base model which always accounts for seasonality.

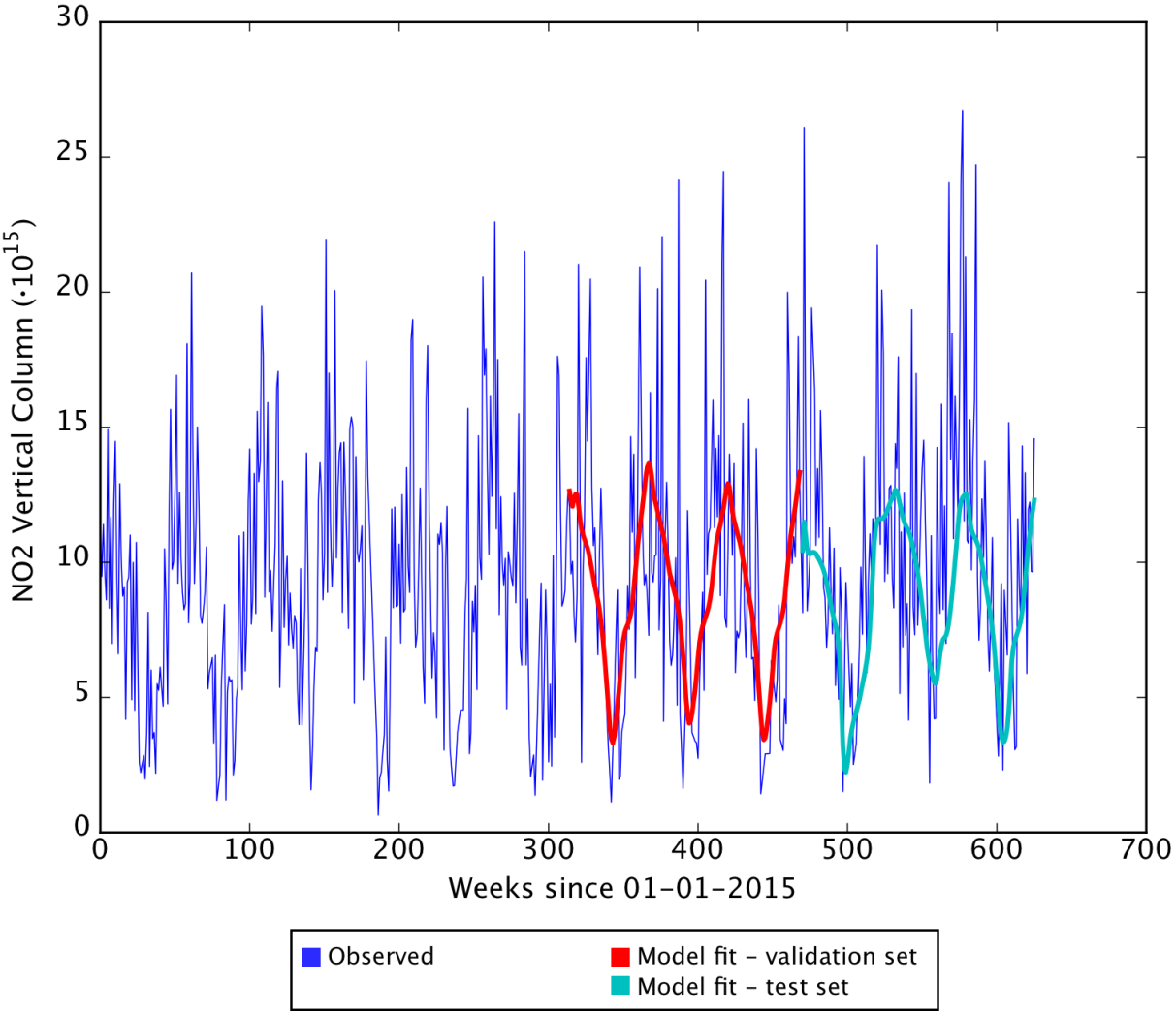

Figure 19: Neural network model result at India's largest coal plant

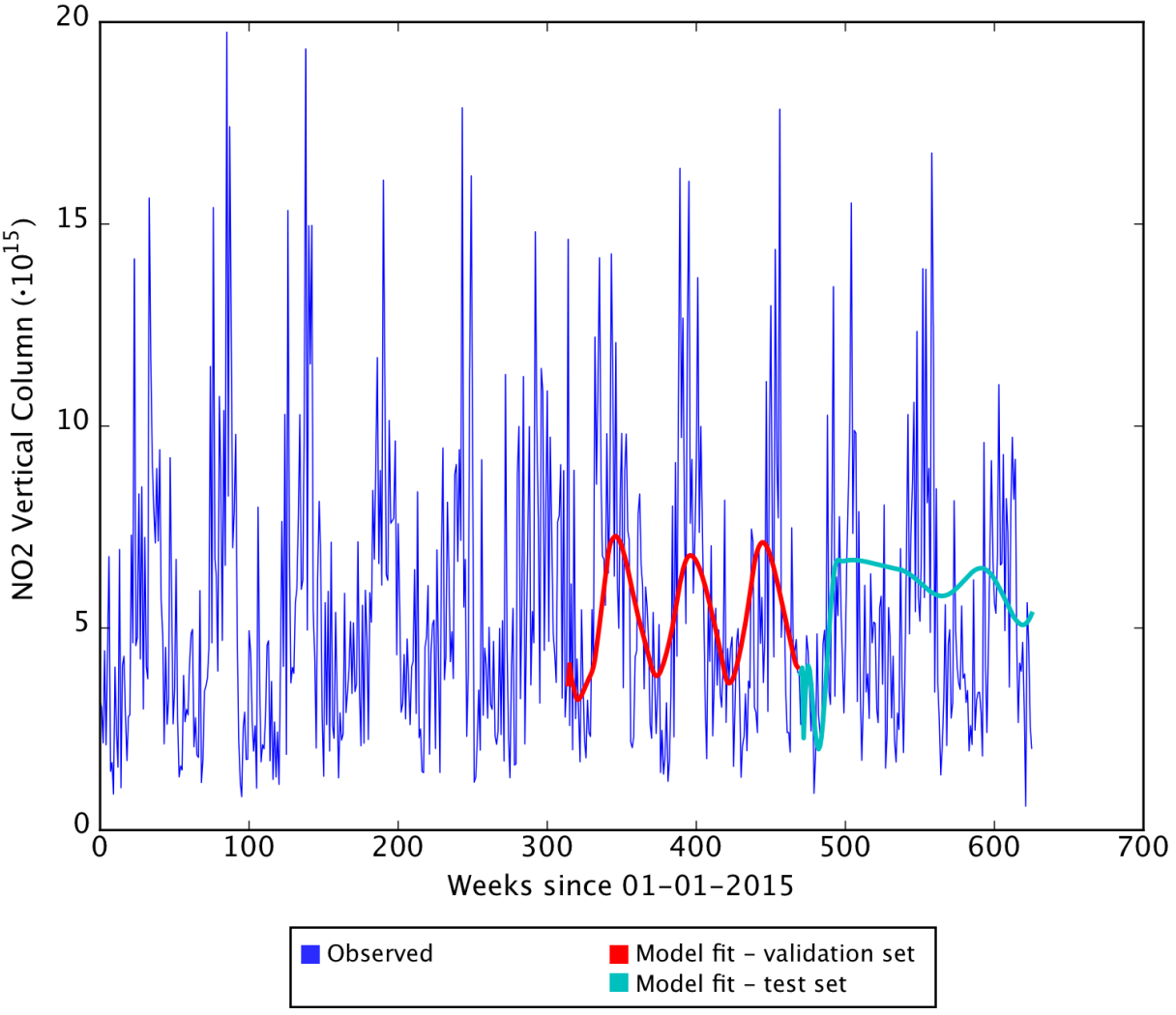

Figure 20: Neural network model result in Riyadh

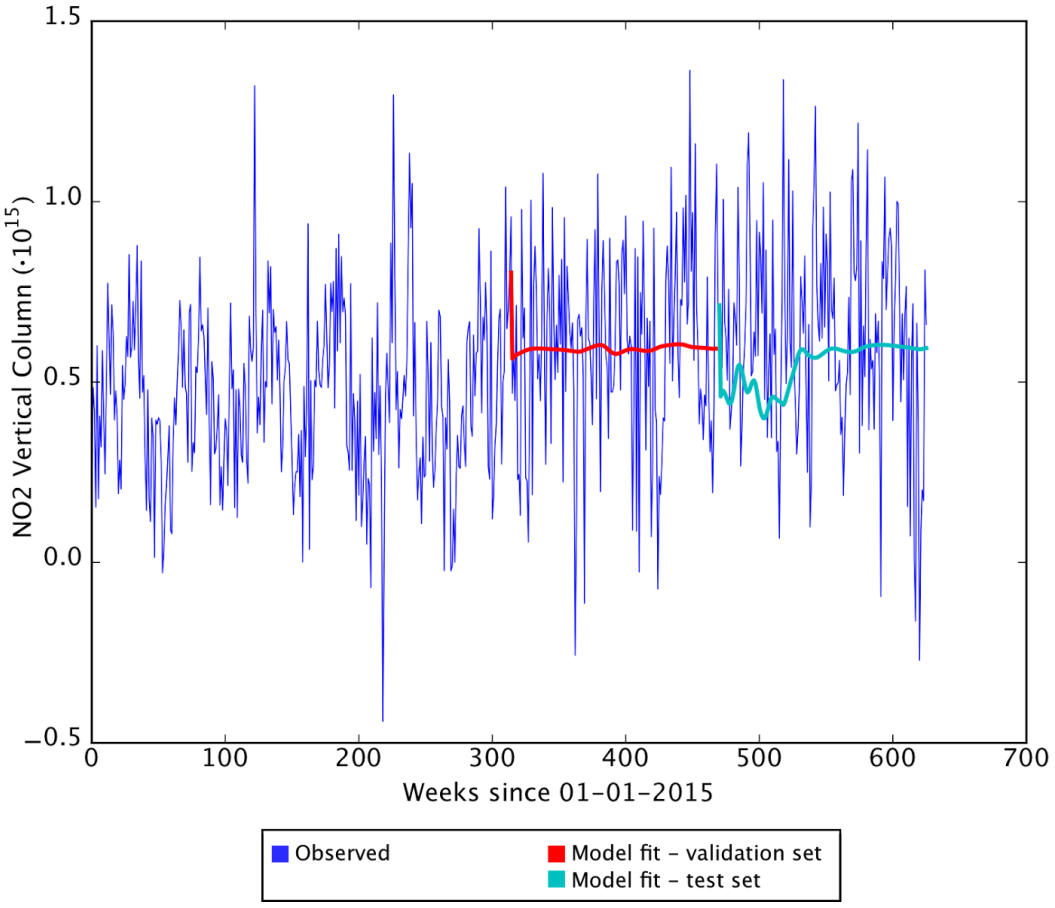

Figure 21: Neural network model result in the Sahara desert

4.2.4 Evaluation

In table 6 below the average, relative, Mean Absolute Errors (MAE) are given for the three areas of interest for the three forecasting methods. Marked in green is the best performing model. We immediately notice that the neural network model does not outperform the other two models in any of the areas. There are multiple possible reasons for this unexpected poor performance.

First, it is possible that the neural network does not have enough data to work with, which makes the neural network unable to find optimal model parameters. Because weekly averages are used for this use case to minimize the amount of missing data points, the total dataset per gridcell consists of only 624 observations[16]. Although the amount of data needed to train a neural network is different for every model and application, in practice it is often in

[16] *12 years · 52 weeks = 624 observations*

the order of tens of thousands to possibly millions of data points. Another indication that the amount of data might be insufficient is that the neural network is not able to match the base model's performance. Given enough data, neural networks are able to predict mathematical functions, such as the sine function. Thus, in theory and given enough data, the neural network should always be able to match the base model's performance by modelling a sine wave with the same parameters as in the base model. That this is not the case in any of the areas of interest, is an indication that the neural network may not have enough data.

Secondly, it is possible that the model configuration or training procedure is not optimal. As discussed above, it is assumed that one model configuration and training procedure is usable for all gridcells. This assumption might be incorrect. Atmospheric systems can be very different in different locations over the earth, which may also cause the need for different approaches.

Area	**Mean**	**Base model**	**ARIMA**	**Neural Net**
India's largest coal plant	5.47	39.3%	41.2%	44.8%
Riyadh	4.31	47.2%	43.7%	45.8%
Sahara desert	0.66	37.8%	40.9%	39.1%

Tabel 6: model performances

The base and ARIMA models perform similar. The three best models in the areas of interest have relative MAE's of 37.8% to 43.7%, which still seems undesirably high. There are two reasons that contribute to these overall high Mean Absolute Errors.

Consider the fact that the actual amount of NO2 present in the atmosphere is mainly decided by two factors: the amount of emissions and meteorological conditions, such as wind and sunlight. The amount of emissions is in turn also partly dependent on the weather, where colder weather generally means more required energy and thus increased emissions.

For this use case, the goal is to make NO2 predictions for up to one year in the future. However, meteorological conditions can only be accurately predicted a few days in advance. The NO2 predictions for the next year, as made in this research, can thus not take these

effects into account. For this reason, we can not expect to estimate the day to day or weekly variation caused by these effects using our method. We can only expect to estimate the longer-term, smoothed trend and seasonal effect. Therefore, it is not surprising that the base model actually approaches the best possible performance considering these circumstances. However, as mentioned above, a neural network should be able to at least match this performance given enough data.

The second factor contributing to the high Mean Absolute Errors is the large uncertainties of the remote sensing data. As discussed in section 3.3, the OMI measurement has a precision of only 35–60% in areas where the NO2 concentration is significant, even lower in areas with lower NO2 concentrations. This also means that a large part of the variance in the time series simply comes from measurement errors. Because of these large uncertainties, the Mean Absolute Error is always expected to be above a certain value for the base model, that fits a smooth curve to the data.

Chapter 5

5 Conclusion

The goal of this research was to help Airbus in taking the next step in the development of the Atmospheric Composition Services platform, by exploring 4 use cases for the platform. These use cases were regridding, air quality forecasting, and change and source detection.

For the first use case of **regridding**, a full and scalable solution was implemented. A regridding algorithm was developed that is entirely independent for each satellite observation, so that it can be fully parallelized using Apache Spark. Using 32 cores, the target processing time of under 1 minute per day was achieved. All data, from October 2004 to present (at the time of writing), was regridded using the algorithm and stored in a Cube data structure. This data cube and its API is subsequently used for the other use cases and by other employees and interns of Airbus.

For the second use case, **air quality forecasting**, three different methods were tested to explore the feasibility and performance of making air quality predictions. Where previous research focused on short term forecasts, we focused on predictions one year in the future. Specifically, the goal was finding out if Deep Learning could be used to make one year predictions with a better performance than simpler regression models. To this end, the performance of Deep Learning models was compared with that of two regression models, a curve fit using nonlinear least squares (base model) and an ARIMA model. This comparison was done by measuring the average performance of the models in three different areas of interest on Earth. We found that Deep Learning for this application did not outperform the other two models. The base model had the best performance.

The contributions of this research can be divided into two parts. First, the code and results of this research contribute to Airbus' development in the ACS platform. The regridding algorithms are continued to be used by Airbus to regrid remote sensing data, as

well as the Cube data structure. Furthermore, the research on air quality forecasting will help Airbus in the further development of this potential commercial service. Lastly, the experiments on the other two use cases in the appendix have helped Airbus gaining more knowledge on the available data and has both provided information and raised questions that can be used and solved by future students that will work in these use cases.

Furthermore, in this research we have shown how a combination of existing methods can be used for the processing and analysis of large amounts of geospatial data. We used the big data processing engine Apache Spark in combination with other libraries, such as Deep Learning and netCDF libraries, towards implementing or prototyping the different use cases. Scalability by parallelization of algorithms was central to this research for every use case examined. This proof of concept can be used by Airbus and future researchers as a starting point when dealing with similar data.

6 Bibliography

[1]	Shi, J. P., & Harrison, R. M. (1997). *Regression modelling of hourly NO x and NO 2 concentrations in urban air in London*. Atmospheric Environment, 31(24), 4081 - 4094.
[2]	Comrie, A. C. (1997). *Comparing neural networks and regression models for ozone forecasting*. Journal of the Air & Waste Management Association, 47(6), 653-663.
[3]	Gardner, M. W., & Dorling, S. R. (1998). *Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences*. Atmospheric environment, 32(14), 2627-2636.
[4]	Rosenblatt, F. (1957). *The perceptron, a perceiving and recognizing automaton* Project Para. Cornell Aeronautical Laboratory.
[5]	Hochreiter, S., & Schmidhuber, J. (1997). *Long short-term memory*. Neural computation, 9(8), 1735-1780.
[6]	Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Mao, M., ... & Ng, A. Y. (2012). *Large scale distributed deep networks*. In Advances in neural information processing systems (pp. 1223-1231).
[7]	Chilimbi, T. M., Suzue, Y., Apacible, J., & Kalyanaraman, K. (2014, October). *Project Adam: Building an Efficient and Scalable Deep Learning Training System*. In OSDI (Vol. 14, pp. 571-582).
[8]	Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., ... & Ghemawat, S. (2016). *Tensorflow: Large-scale machine learning on heterogeneous distributed systems*.

[9]	Boersma, K. F., Eskes, H. J., & Brinksma, E. J. (2004). *Error analysis for tropospheric NO2 retrieval from space*. Journal of Geophysical Research: Atmospheres, 109(D4).
[10]	Box, G. E., Jenkins, G. M., Reinsel, G. C., & Ljung, G. M. (2015). *Time series analysis: forecasting and control*. John Wiley & Sons.
[11]	Samuel, A. L. (1959). *Some Studies in Machine Learning Using the Game of Checkers*. IBM Journal of Research and Development, vol. 3, no. 3, pp. 210-229, July 1959.
[12]	Gollapudi, S. (2016). *Practical Machine Learning*. Packt Publishing Ltd. pp 21-22.
[13]	Bottou, L. (1991). *Stochastic Gradient Learning in Neural Networks*. Proceedings of Neuro-Nîmes 91.
[14]	Priddy, K. L., & Keller, P. E. (2005). *Artificial neural networks: an introduction* (Vol. 68). SPIE press. pp. 45.
[15]	Brockwell, P. J., & Davis, R. A. (2016). *Introduction to time series and forecasting*. Springer.
[16]	Ng, A. (2015). *What data scientists should know about Deep Learning*. ExtractConf 2015. Retrieved from: https://www.youtube.com/watch?v=O0VN0pGgBZM
[17]	Längkvist, M., Karlsson, L., & Loutfi, A. (2014). *A review of unsupervised feature learning and deep learning for time-series modeling*. Pattern Recognition Letters, 42, 11-24.
[18]	Beirle, S., Boersma, K. F., Platt, U., Lawrence, M. G., & Wagner, T. (2011). *Megacity emissions and lifetimes of nitrogen oxides probed from space*. Science, 333(6050), 1737-1739.

[19]	Beirle, S., Platt, U., Wenig, M., & Wagner, T. (2003). *Weekly cycle of NO 2 by GOME measurements: A signature of anthropogenic sources.* Atmospheric Chemistry and Physics, 3(6), 2225-2232.

Chapter 7

7 Experimentation

In the context of this research several other use cases for the Atmospheric Composition Services platform have been identified. For two of these use cases, some experiments have been done to explore the feasibility and difficulties for these use cases. In the future, Airbus DSNL will take on other students that will research these use case more in-depth.

7.1 Source detection

For this use case, the goal is to detect and identify emission sources using satellite data. Experiments using two approaches have been performed for this use case in the context of this research.

7.1.1 Peak finding

In this approach a peak finding algorithm was applied to NO2 yearly averages to find candidate emission sources. NO2 will move along with the wind. Thus, the locations of the NO2 concentrations observed by the satellite do not match the locations of the actual emission source. The idea is that by averaging the NO2 over one year this impact of the wind will average out.

The peak finding algorithm considers all gridcells. A gridcell is a candidate emission source if:

1. its value is the maximum in an area of size ***A*** around itself, and
2. its value is higher than a threshold ***t***, and
3. its value is more than ***r*** times higher than the local average, which is calculated over an area of size ***B*** around itself.

The rationale behind the absolute threshold is that there are many, small local maxima in areas with low NO2 concentrations due to noise in the satellite data, even when taking yearly averages. We want to ignore these cells. The rationale behind the third rule is similar, but by setting a threshold relative to the local average around the gridcell we can set the absolute threshold of the second rule lower. This way we do not lose the ability to detect smaller sources.

7.1.2 Including wind data

Of course, in the approach above, the averaging out will only work correctly if the wind direction is completely random over the year, which in a lot of locations is not the case. For example the Netherlands has a prevailing southwest wind, which means that on average the wind will blow from the southwest.

For this reason, in this approach the satellite data is combined with wind data from the European Centre for Medium-Range Weather Forecasts (ECMWF). ECMWF has publicly available datasets for many meteorological parameters on a global level. In this approach the parameters that are used are the wind's *u* and *v* vector components. This data is downloaded from ECMWF in a grid with cells of 0.125 by 0.125 degrees. With this wind data two separate experiments were performed.

For the first experiment, it was explored if we can transform the NO2 concentrations using the wind data, theoretically 'moving' the NO2 back to where it was emitted. This was done by modifying the regridding algorithm from section 4.1, so that for every single satellite observation the corresponding wind vector based on time and location was taken, and the satellite observation was transformed along the negative of the wind vector. However, there was too much missing information and thus too many (bad) assumptions in doing this transformation and therefore the results were not very useful. For example, information on how long the observed NO2 is in the atmosphere, or if the observed NO2 is near the surface or higher in the atmosphere (i.e. the *vertical profile*).

For the second experiment, the approach was to take year averages, but only include the NO2 observations that were made when the windspeed in that location was below a certain limit $\boldsymbol{w}$, similiar to Beirle et al. (2011) [18]. By only taking all observations with a very low windspeed, the observed NO2 will be close to the source of the emitted NO2.

To this end, a data cube was created with four dimensions: latitude, longitude, time and windspeed. The windspeed dimension contains 10 cumulative levels in the range [0,9]. Level 1 contains the data on the observations with a windspeed between 0.0 and 1.0 m/s, level 2 the count between 0.0 and 2.0 m/s, etcetera. The last level, Level 10 contains the observation count without a wind restriction. With this cube, the data for different windspeed limits $\boldsymbol{w}$ can be easily retrieved.

7.1.3 Number of observations

Because for the approach above some observations would be filtered out, the question rose how many observations we would have left, and consequently how many observations we had at all for each location when taking monthly or yearly averages. It was found that the amount of observations varies heavily in both space and time, which was to be expected. However, for some months the amount of observations was a lot less than expected. See for example India (left) and Ireland (right) in July 2010 below in figure 22. Everything not dark green has < 4 observations in the entire month, with large areas even having 0-1 observation. Note that no limit was put on the windspeed.

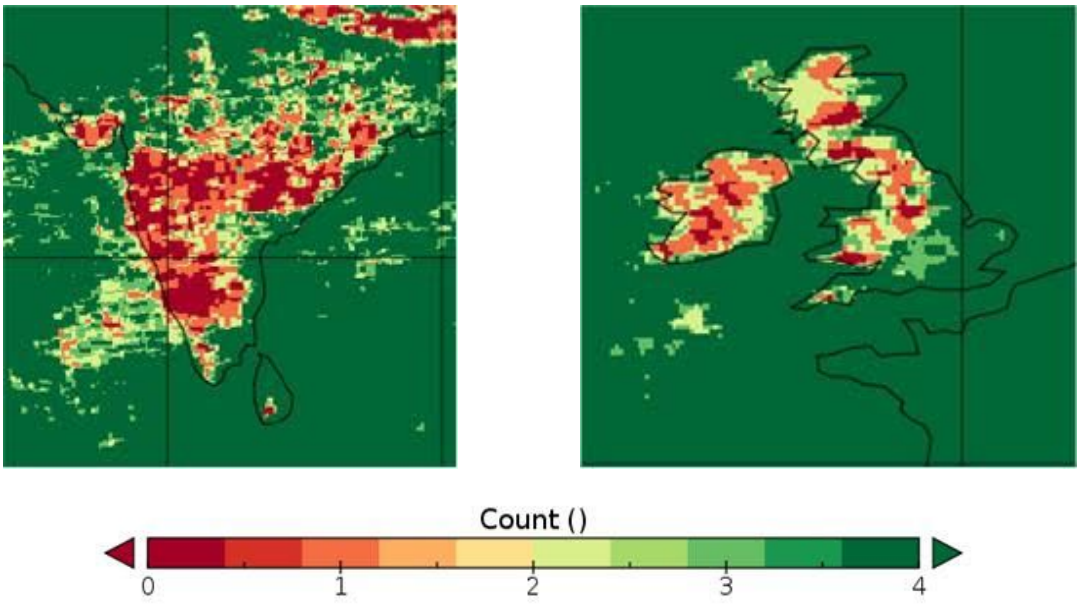

Figure 22: number of observations for India and Ireland/United Kingdom

No limit on wind

Concluding, on a per-project basis there should be looked at the amount of available observations for the region of interest to see what time resolution can be achieved with a reasonable amount of observations. This should be done per project because it is so dependent on the region/time.

7.2 Change detection

The goal for this use case is to detect changes in the atmospheric composition. We identify two types of changes, momentary changes and long-term changes.

7.2.1 Momentary changes

For momentary changes the goal is to detect significant changes in as short a period as possible. The cause of such a change could for example be a forest fire or the turning on of a coal plant. The main problem that has to be solved for this use case is finding a test of significance that is valid for the available data, which is noisy and has large uncertainties.

For the OMI data products, the retrieval uncertainties are estimated and included in the data product. This estimation is done on a pixel-to-pixel basis, following the method as published by Boersma et al. (2004) [9]. This error can be used in the change detection, considering each observation as a separate distribution with a mean (i.e. the observed value) and the standard deviation (i.e. the quantified OMI error).

For this experiment two-weekly averages are used. The hypothesis in this test is that the difference in the means of two consecutive two-weekly averages is zero. Using a *two-sample t-test* a 95% confidence interval is constructed for this difference in means. This confidence interval is given by:

$$(\overline{x_1} - \overline{x_2}) \pm t^* \sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}$$

Although for a two-sample t-test the t-statistic does not exactly follow the t-distribution, its value can be conservatively estimated by taking *t(k)*, where *k* is $n_1 + n_2 - 2$. This value *t(k)* can be looked up in the table with t-values below. In this table the value *k* is denoted with *df*. For a 95% confidence interval, the values in the column *0.025* are used.

df	0.4	0.25	0.1	0.05	0.025	0.01	0.005	0.0005
1	0.32492	1	3.077684	6.313752	12.7062	31.82052	63.65674	636.6192
2	0.288675	0.816497	1.885618	2.919986	4.30265	6.96456	9.92484	31.5991
3	0.276671	0.764892	1.637744	2.353363	3.18245	4.5407	5.84091	12.924
4	0.270722	0.740697	1.533206	2.131847	2.77645	3.74695	4.60409	8.6103
5	0.267181	0.726687	1.475884	2.015048	2.57058	3.36493	4.03214	6.8688
6	0.264835	0.717558	1.439756	1.94318	2.44691	3.14267	3.70743	5.9588
7	0.263167	0.711142	1.414924	1.894579	2.36462	2.99795	3.49948	5.4079
8	0.261921	0.706387	1.396815	1.859548	2.306	2.89646	3.35539	5.0413
9	0.260955	0.702722	1.383029	1.833113	2.26216	2.82144	3.24984	4.7809
10	0.260185	0.699812	1.372184	1.812461	2.22814	2.76377	3.16927	4.5869
11	0.259556	0.697445	1.36343	1.795885	2.20099	2.71808	3.10581	4.437
12	0.259033	0.695483	1.356217	1.782288	2.17881	2.681	3.05454	4.3178
13	0.258591	0.693829	1.350171	1.770933	2.16037	2.65031	3.01228	4.2208
14	0.258213	0.692417	1.34503	1.76131	2.14479	2.62449	2.97684	4.1405
15	0.257885	0.691197	1.340606	1.75305	2.13145	2.60248	2.94671	4.0728
16	0.257599	0.690132	1.336757	1.745884	2.11991	2.58349	2.92078	4.015

When this confidence interval does not include 0, i.e. the lower bound is higher than zero or the upper bound of the interval is lower than zero, the means are considered significantly different from eachother and thus a significant change has been detected. An example result for the two-sample t-test is given in figure 23. The blue line depicts the two-weekly average NO2 vertical column. Red dots denote significant decrease, green significant increase compared to previous two-weeks.

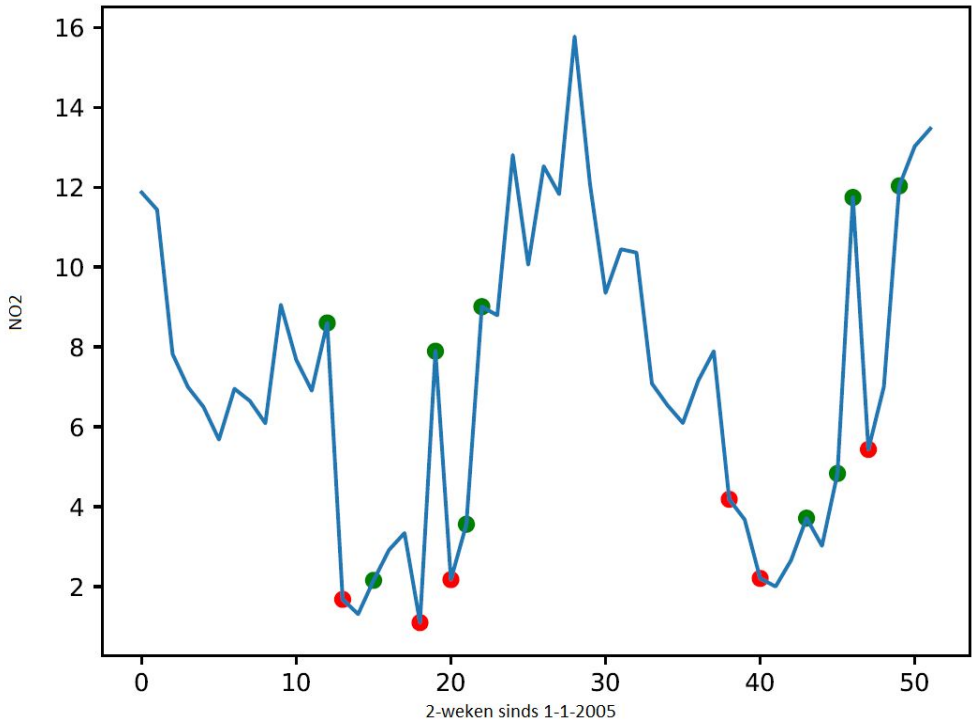

Figure 23: Example result of two-sample t-test.

However, the assumptions for the two-sample t-test are problematic. The most fundamental problem with the data and the assumptions is that this test assumes equal variances between the two populations. However, the quantitative error in the OMI data is correlated with the observation itself. In other words, a higher NO2 will have a higher error. Of course, with significant changes one observation is always significantly higher than the other, thus having a significantly higher error. Therefore, the two populations for a significant change will never have the same variance.

This last problem with the OMI errors is not only problematic for the t-test, but also in general for the detection of significant changes. Finding a way to deal with the high uncertainty of the data in this use case falls outside the scope this research and is left to the next student that will research this use case in more detail.

7.2.2 Long-term

In the experiments for this use case there was also briefly looked at long-term changes. To this end, a linear fit was for all gridcells, using 11 years of data between 2005 and 2016. This linear fit is of the form:

$$y = ax + b$$

In figure 24 below, the coefficients a are depicted for the entire globe. The linear fit was done through weekly averages, which means that a coefficient $a = 0.005$ can be interpreted as the NO2 vertical column increasing on average with $0.005*10^{15}$ particles NO2 per week.

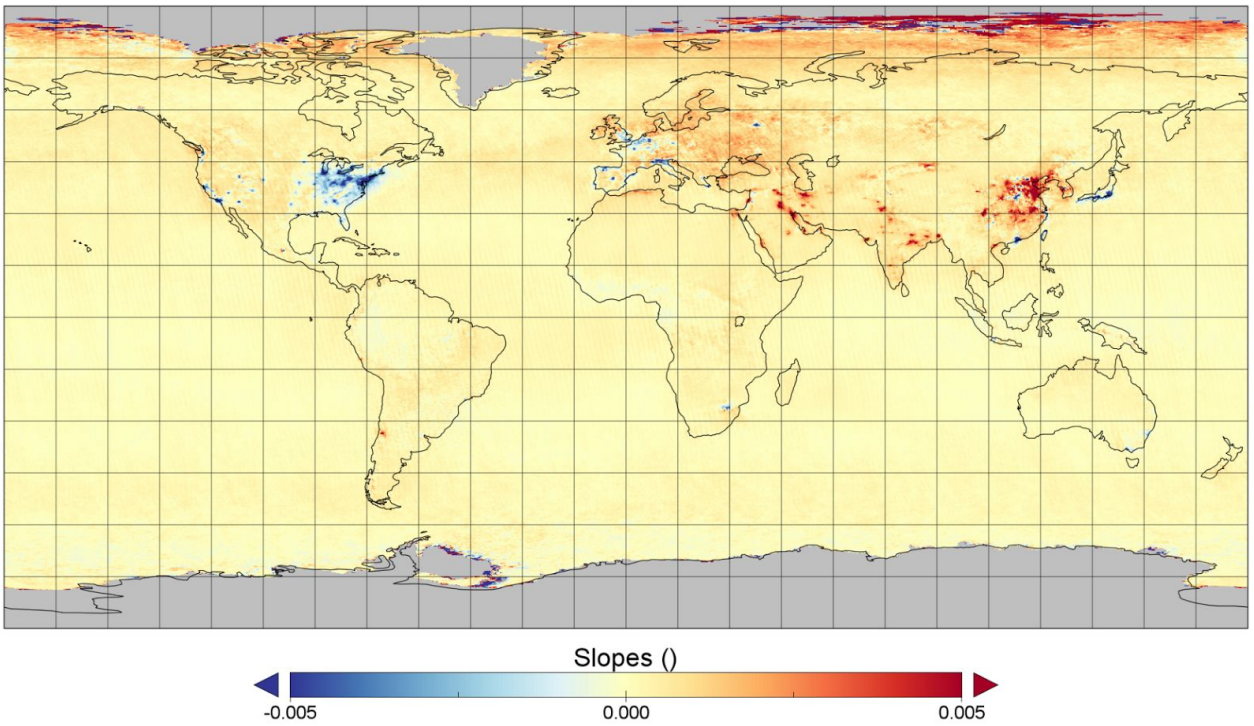

Figure 24: Coefficients *a* from linear fit

This linear fit is very interesting and useful to see the long-term trend for NO2 for every location on the earth. We can see that in the western world the emission of NO2 has declined over the last 11 years, whereas in other countries, most importantly India and China, the emission has increased.

However, for detecting long-term trend *changes,* as is the goal in this use case on change detection, future research has to be done in applying *structural break detection* on the data. Break detection allows detection of changes in the trend, for example analysis on the long-term effect of the introduction of extra regulations.