GRAZ UNIVERSITY OF TECHNOLOGY

MASTER'S THESIS

Linear Regression and Artificial Neural Networks for Parameter-Modeling of a 4G Link for Unmanned Aerial Vehicles

Giancarlo Benincasa

supervised by
Prof. Erich LEITGEB
Klaus KAINRATH, PhD

March 16, 2021

Deutsche Fassung:
Beschluss der Curricula-Kommission für Bachelor-, Master- und Diplomstudien vom 10.11.2008
Genehmigung des Senates am 1.12.2008

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

Graz, am ……. ……..
(Unterschrift)

Englische Fassung:

STATUTORY DECLARATION

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

……. ……..
date (signature)

Danksagung

Mein Dank gilt meinem Vater, der leider nicht mehr unter uns weilt, meiner Mutter, die mich immer unterstützt hat, auch wenn es (bzw. ich) nicht immer ganz einfach war, meinen Tanten, meiner Oma und meinem Opa, der früh mein Interesse für Technik weckte und ohne den ich wahrscheinlich niemals die Ingenieurslaufbahn eingeschlagen hätte. Auch den italienischen Zweig meiner Familie möchte ich dankend erwähnen, auch wenn ich euch leider zu selten besuchen kann. (Fast) jedes Jahr kann ich bei euch im Urlaub viele positive Impulse mitnehmen, die mein Leben bereichern und mir helfen, in schwierigen Zeiten durchzuhalten und mich nach einem anstrengenden Jahr zu erholen.

Weiters möchte ich meinen Betreuern Klaus Kainrath und Erich Leitgeb danken. Am IHF habe ich mich immer wohl und willkommen gefühlt, was bei der Anonymität auf der Uni, insbesondere im Bachelor, leider viel zu selten vorkommt. Lobend hervorheben möchte ich auch, dass ihr mir bei der Wahl der Methoden bzw. der Herangehensweise freie Hand gelassen und mir somit einen Vertrauensvorsprung gegeben habt, der hoffentlich nicht enttäuscht wurde.

Auch danken möchte ich meinen Freunden und Kollegen, von denen gerade jetzt, im Augenblick, in dem ich diese Sätze schreibe, einige meine Masterarbeit lesen, um mir mit der Korrektur zu helfen. Hoffentlich können wir uns bald wieder öfter sehen!

Von meinen Freunden möchte ich insbesondere Felix und Sophie dankend erwähnen. Wir waren in den letzten beiden Jahren ein wirklich gutes Team! Ich hoffe (und da bin ich sehr zuversichtlich), dass sich auch unsere beruflichen Laufbahnen in Zukunft kreuzen werden.

Und zu guter Letzt möchte ich meiner Freundin Alina danken, mit der ich gerade Seite an Seite diese schwierige (und hoffentlich bald endende) Zeit überstehe. Du bist die beste.

Kurzfassung

Diese Masterarbeit entstand im Zuge des Forschungsprojekts "Evaluierung von Kommunikationstechnologien für den Betrieb von Unmanned Aerial Vehicles innerhalb und außerhalb des Sichtbereichs". Dieses stellt einen Teil der Dissertation von Klaus Kainrath dar.

Die bisherige Gesetzgebung für den Betrieb von Unmanned Aerial Vehicles (UAVs) regelte die Bedingungen auf nationalstaatlicher Ebene. Die 2021 inkraftgetretenen EASA-Regularien sollen diese Regelungen EU-weit vereinheitlichen, wozu Kompromisse zwischen Betriebssicherheit und Wirtschaftlichkeit gefunden werden müssen. Für Ersteres sind zuverlässige Datenlinks zwischen UAVs und deren Betreibern notwendig, wozu sich die bestehende 4G-Netzinfrastruktur anbietet, insbesondere im Betrieb außerhalb des Sichtbereichs (Beyond Visual Line of Sight oder BVLOS).

Die vorliegende Arbeit untersucht mit linearer Regression und künstlichen neuronalen Netzen zwei Technologien, die die Modellierung der relevanten Empfangsqualitätsparameter RSRP (Reference Signal Received Power) und RSRQ (Reference Signal Received Quality) in Abhängigkeit der Position des UAV ermöglichen sollen.

Hinsichtlich der linearen Regression wird ein Vergleich zu bestehenden Modellen, welche den Empfang in Bodennähe modellieren, gezogen und eine Erweiterung, die diese Vorgehensweise für den Betrieb in größeren Höhen geeignet machen soll, vorgeschlagen.

Mit dem Aufkommen immer größerer analysierbarer Datenmengen in den letzten Jahren und Jahrzehnten begann auch der Siegeszug der künstlichen neuronalen Netze, deren Grundlagen bereits in den 1940er Jahren erforscht wurden. In der heutigen Zeit sind sie definitiv eines der Hot Topics in Technik und Wissenschaft. Weniger bekannt ist hingegen, dass sich vor allem relativ simple Feedforward Neural Networks sehr gut für Regressionsprobleme eignen, wozu oft auch relativ kleine Datenmengen genügen. Dies liegt vor allem daran, dass die Approximation beliebiger Funktionen mit nur einer Schicht von versteckten künstlichen Neuronen möglich ist. In dieser Arbeit wurde der Versuch unternommen, ein Modell für die Parameter RSRQ/RSRP in Abhängigkeit der Position des UAV mit relativ wenigen Schichten und Trainingsdaten zu erschaffen.

Die Ergebnisse dieser Arbeit zeigen, dass eine ausreichende Verbindungsqualität im 4G-Netz, zumindest in den niedrigeren Frequenzbändern, zu erwarten ist, auch wenn aufgrund der bodennahen Ausrichtung der Sendeantennen in der Regel reflektierte Signale (non line of sight, NLOS) anstatt direkter Komponenten (line of sight, LOS) empfangen werden. Wie sich diese Aspekte bei höheren Frequenzen und vor allem im kommenden 5G-Netz verhalten, ist noch zu erforschen.

Die Ergebnisse der Modellierung mittels neuronaler Netze zeigen das Potential dieser Technologie, auch wenn sich die Prädiktionsmöglichkeiten (noch) auf Bereiche in der Nähe von Messpunkten beschränken. In der Zukunft könnten komplexere Strukturen trainiert werden, die auch andere Parameter als nur die Position als Eingangsvariablen verwenden, um zum Beispiel Frühwarnsysteme für Funklöcher zu entwickeln.

Abstract

This master thesis is a result of the research project *"Evaluierung von Kommunikationstechnologien für den Betrieb von Unmanned Aerial Vehicles innerhalb und außerhalb des Sichtbereichs"*[1]. This project is a part of the dissertation of Klaus Kainrath.

Previous legislation for Unmanned Aerial Vehicle (UAV) operation was based on national laws, which will change as new EASA regulations come into effect 2021, which are unified across the entire EU. To find proper legislation for this field, trade-offs between operational safety and economic viability must be found. For the former aspect, reliable data links between UAVs and their operators are necessary, which may be offered by the existing 4G net infrastructure, especially under beyond visual line of sight (BVLOS) conditions.

With linear regression and artificial neural networks, this thesis aims to conduct investigations on technologies, which enable modelling of the relationship between the UAV's position and the relevant reception quality parameters RSRP (Reference Signal Received Power) and RSRQ (Reference Signal Received Quality).

Regarding linear regression, a comparison to existing models, which are valid for signal reception in ground proximity, is made. Furthermore, an extension that enables this approach for greater UAV heights is suggested.

With the advent of ever increasing amounts of analyzable data in recent years and decades, the use of artificial neural networks, for which the basics were actually already developed in the 1940s, has become more widespread. Today, they are definitely one of the hot topics in technology and science. What is less known however is the fact that even relatively simple feedforward neural networks are well suited to solve regression problems, for which often even small amounts of data suffice. This is for the most part true because the approximation of arbitrary functions is possible with just one hidden layer of artificial neurons. In this thesis, an attempt was made to create a model for the reception quality parameters RSRP/RSRQ as a function of the UAV's position with a relatively small amount of hidden layers and training data.

The results show that a sufficient reception quality is to be expected in the Long Term Evolution network, at least in it's lower frequency bands. This is true even though usually non line of sight (NLOS) signals are received by the user equipment instead of line of sight (LOS) components due to the transmitter antennas being oriented such that receivers in ground proximity are served. How these aspects are to be valued for higher frequencies, and especially for the upcoming 5G network, is still to be investigated.

The results of the modelling via neural networks show the potential of this technology, even though the prediction possibilities are confined to areas close to known measurement points, at least up until now. Future research and development may lead to training of more complex structures, which also take parameters other than just the position as input variables. This could for example be used to build early warning systems for dead zones.

[1] translates to "Evaluation of Communication Technologies for the Operation of Unmanned Aerial Vehicles within and beyond Visual Line of Sight"

Contents

1 Introduction

The first chapter serves as a short description of the aims of this thesis and the means that they were tried to be reached with. Furthermore, the relevant parameters Reference Signal Received Power (RSRP), Reference Signal Received Quality (RSRQ) and Received Signal Strength Indicator (RSSI) are described. These determine the reception quality of a Long Term Evolution (LTE) wireless link.

1.1 Goals

The goals of this thesis were to find models which enable a prediction of the cellular reception quality as a function of the user equipment's relative position to the transmitter station. The emphasis was put on measurements conducted in relatively high altitudes since the main goal of the project is to determine to which degree unmanned aerial vehicles (UAVs) can be controlled via mobile radio and to help finding proper legislation. There were two main investigations conducted:

Linear regression for RSRP/RSRQ

Linear regression was used to find a model for path loss as a function of the distance between transmitter station and mobile device, based on RSRP measurements. Furthermore, a linear regression analysis for RSRQ was conducted.

Prediction of RSRP and RSRQ via Machine Learning

Deep neural networks were used to model the relationship between the position of the mobile device relative to the transmitter station and the RSRP and RSRQ parameters.

All measurements were conducted in rural areas since Austrian legislation regarding UAVs is very strict when it comes to operation in urban areas. This has to be taken into account since there are significant differences between rural and urban areas when it comes to wave propagation. The former usually enable a visual line of sight connection, while the latter usually don't (beyond visual line of sight - BVLOS). The aim of this thesis is to contribute to enabling future economic use of UAVs, which are controlled via mobile radio technologies like the fourth generation mobile radio technology LTE (Long Term Evolution).

1.2 Methods

The following section serves as a short overview of the two main methods for modelling: Linear regression for mobile radio parameter (RSRP/RSRQ) prediction depending only on the range (and the elevation angle between base station and user equipment) and neural networks for parameter prediction in 3D coordinates.

1.2.1 Linear regression

As already mentioned, the main modeling methods were linear regression and deep neural networks. The former can only deliver meaningful results if the elevation and azimuth angles between base station (BS) and user equipment (UE) are kept at a constant value since these heavily influence RSRP/RSRQ. Deep neural networks on the other hand can become more intelligent models and take either Cartesian or polar

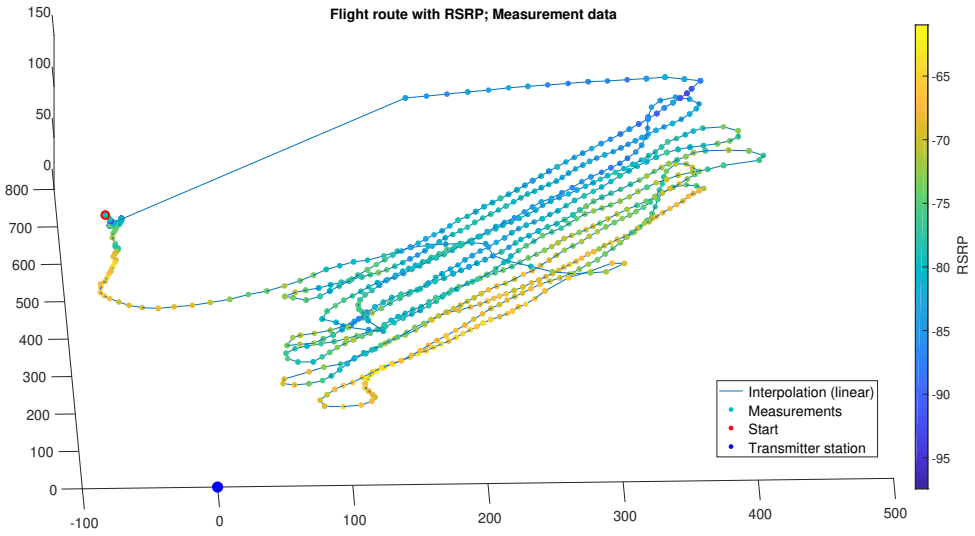


Figure 1: One of the original flight datasets for linear regression, axes in meters

coordinates as input and map these to RSRP or RSRQ values. Thus, the input data needed for these approaches are different: While linear regression needs input values with fixed angle values, machine learning profits from a wide variety of input data combinations to be able to learn better.

Figures 1 and 2 contain one of the flights and one of the corridors that were selected from that flight and evaluated. φ denoted the azimuth angle and θ the elevation angle. The former is usually kept in an interval of about 3 degrees to make sure that no useful data is deleted. The boundaries for the elevation angles are changed in one degree steps, which still gives us enough data for linear regression in any given interval, as can be seen in figure 2.

Calculating the linear regression $y = k \cdot x + d$ with MATLABs integrated command (mean square error cost function) leads to the result depicted in figure 3. It is clearly visible that linear regression enables a proper modeling of path loss in this scenario. There is some variance around the best fitting line, which is mostly due to multipath components interfering with the line of sight component destructively or constructively.

Path loss was calculated for multiple beams with the upper and lower limit of θ incremented both in one degree steps and a fixed φ range. Then the average was calculated, weighted by the number of measurements in any given beam to retrieve an estimate for the path loss. Figure 4 shows the result of one example analysis. The marked data point has an elevation angle of 5.5°, which means that the beam actually includes all measurements, for which $\theta \in (5°, 6°)$.

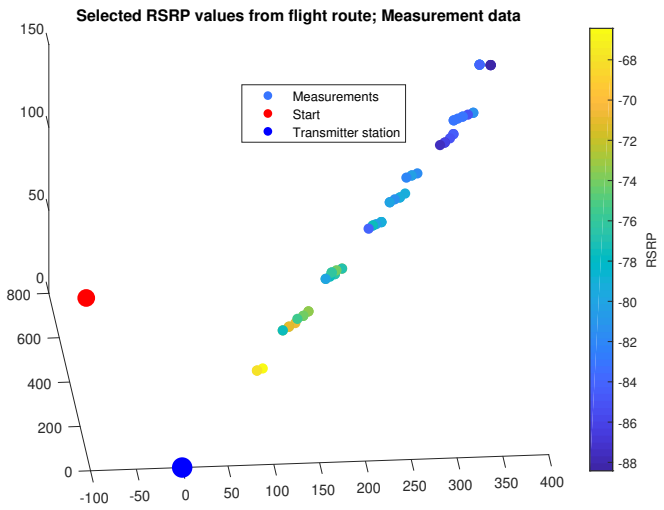


Figure 2: Selected RSRP values from original flight data: $\varphi \in (59°, 62°), \theta \in (10°, 11°)$

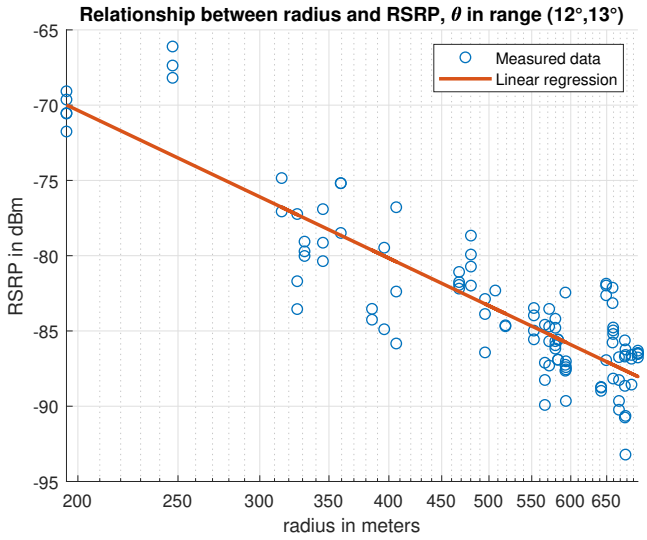


Figure 3: Selected RSRP values from one of the original flight datasets: $\varphi \in (73°, 76°), \theta \in (12°, 13°)$

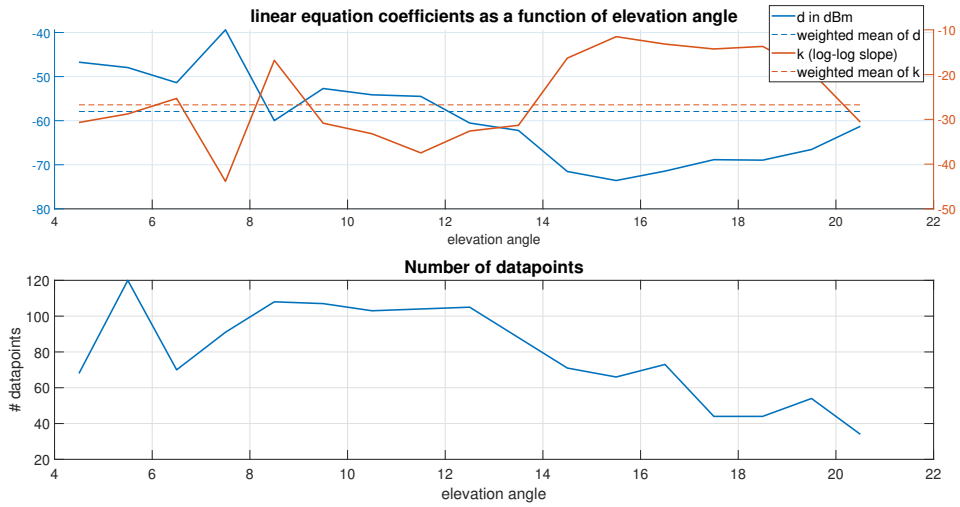


Figure 4: One example for the results of multiple beams: $\varphi \in (73°, 76°), \theta \in (4°, 21°)$

1.2.2 Neural networks

As already mentioned, neural networks profit from a wide variety of input data. Thus, a typical flight route for training does not exhibit the form of a wall, but is rather chosen such that many different combinations of radii and angles occur, see figure 5.

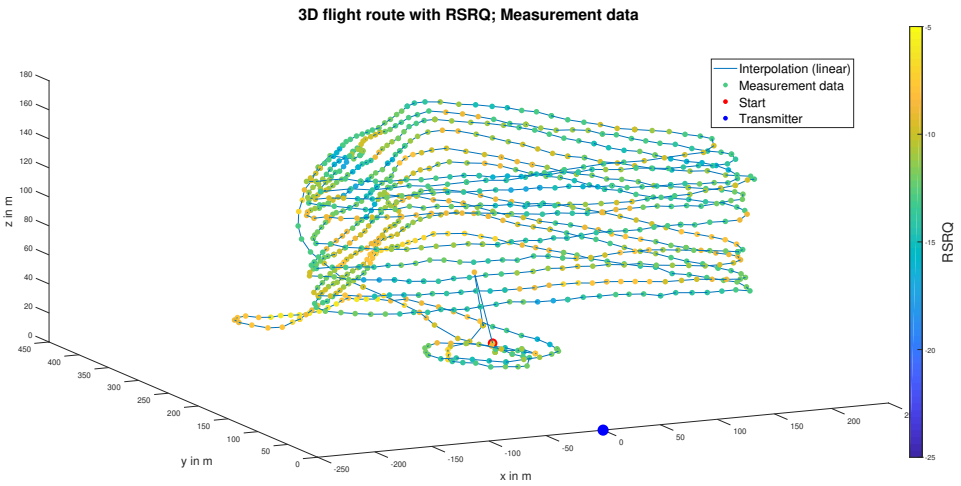


Figure 5: A typical flight route for deep learning. Input data can now also be RSRQ.

The inner workings of artificial neural networks will be discussed in chapter four. Since MATLAB is not the Author's first choice for machine learning, these parts of the thesis are implemented in python, using Googles Tensorflow library. For the regression task at hand, a relatively simple feedforward neural network was used. Best performance was achieved with a depth of four layers, but also experiments with more layers were conducted.

1.3 Reception quality parameters

The following section describes the commonly used reception quality parameters RSRP, RSRQ and RSSI. Due to the real world conditions that the measurements were conducted in, an actual measurement of the link budget is not possible. However, the parameters serve as a viable approximation. RSRP is actually used to estimate path loss in mobile radio devices. Under practical use conditions, the reception quality (i.e. Bandwidth, "smoothness" of latency etc.) is mainly determined by the values of RSRP and RSRQ.

1.3.1 RSRP - Reference Signal Received Power

One of the most important reception parameters is the reference signal received power. The time-frequency diagram in figure 6 shows two LTE resource blocks, which contain four reference signals each. They are transmitted at a fixed power, which is often higher than other resource elements and are used to approximate the path loss between transmitter and receiver. The reference signals transmit a complex value (useful for channel estimation) that depends on their positioning within the resource block and the transmitting cell. However, they have to be inserted at the first and third last OFDM symbol, with a frequency spacing of six sub-carriers. The RSRP value is determined by the linear average of the received reference signal power over N reference signals:

$$RSRP[W] = \frac{1}{N} \sum_{n=1}^{N} P_{rs,n} \tag{1}$$

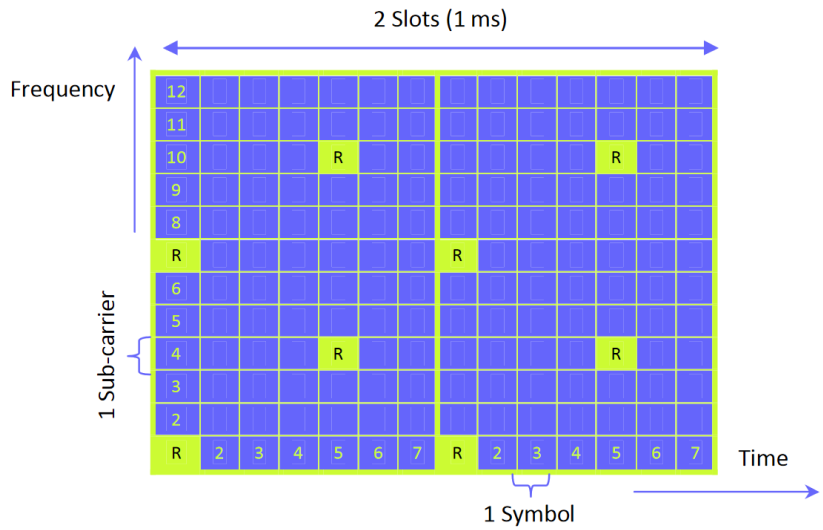


Figure 6: Depiction of two LTE resource blocks, i.e. one subframe. The yellow resource elements marked with R are the reference signals [1].

Per definition [2], the reference point of the RSRP is the antenna connector of the user equipment. Furthermore, if antenna diversity is used, the reported RSRP value shall not be lower than the measured value of any of the diversity branches.
The reference signal received power alone is no testimony to the received signal quality since these mea-

surements also pick up interference in the respective frequency ranges and need to be set in relation to the received signal strength indicator (RSSI) to get the full picture.

1.3.2 RSSI - Received Signal Strength Indicator

The RSSI serves more or less as an overview of the power in the used spectrum by the channel, as it simply measures the complete power contained in it [4]. Thus, it basically measures the noise floor, including thermal noise in the receiver and interference from neighboring cells plus the useful power that is received. If the transmission bandwidth is higher (i.e. more channels are used), a higher RSSI is to be expected as it does not measure a power spectral density, but the total power. Thus, the number of resource blocks needs to be considered when calculating the RSRP.

1.3.3 RSRQ - Reference Signal Received Quality

A very important aspect for practical reception quality is the reference signal received quality [5]. It consists of the ratio between RSRP and RSSI, multiplied by the number of resource blocks used for RSRP calculation N (depends on the bandwidth of the channel):

Channel Bandwidth = 1,4 MHz:	7 resource blocks (currently not used)
Channel Bandwidth = 5 MHz:	25 resource blocks (currently not used)
Channel Bandwidth = 10 MHz:	50 resource blocks (e.g. LTE 800)
Channel Bandwidth = 15 MHz:	75 resource blocks (e.g. LTE1800 in areas with GSM1800)
Channel Bandwidth = 20 MHz:	100 resource blocks (e.g.. LTE1800 and LTE 2600)

The actual RSRQ value is calculated as:

$$RSRQ = N \cdot \frac{RSRP[W]}{RSSI[W]} \tag{2}$$

The values of RSRP and RSSI are put into the equation as Watt or mW-values. Since the RSRQ is a ratio of powers, it has no dimension and is usually written logarithmically.

$$RSRQ_{dB} = 10 \cdot log_{10}(RSRQ) = 10 \cdot log_{10}\left(N \cdot \frac{RSRP[W]}{RSSI[W]}\right) = 10 \cdot log_{10}(N) + RSRP_{dBm} - RSSI_{dBm} \tag{3}$$

1.3.4 Reception quality estimates

As already mentioned previously, mainly RSRQ and RSRP influence the perceived reception quality, influencing bandwidth, latency and stability. A rule of thumb for RSRP is that values higher than -100 dBm are decent and for values up to -113 dBm, the signal is dead. Below, a table including perceived reception quality and respective parameter value ranges can be found. ([1], [5])

Perceived connection quality	RSRP range
excellent - no problems whatsoever	-50 ... -65 dBm
good connection conditions	-65 ... -80 dBm
not perfect, but good enough for stable connection	-80 ... -95 dBm
acceptable conditions, short outages may occur	-95 ... -105 dBm
very weak reception, measures must be taken	-110 ... -125 dBm
extremely bad, probably no connection possible	less than -125 dBm

For excellent connection properties, usually there has to be a line of sight connection. Theoretically, RSRQ can exhibit any value. However, physical limitations lead to a range of

$$-3dB < RSRQ < -20dB \tag{4}$$

Influence of noise and interference	RSRQ range
optimum connection quality, no influence by interference	-3 dB
interference exists but has no effect	-4 ... -5 dB
interference has an influence on connection quality	-6 ... -8 dB
connection quality notably influenced by interference	-9 ... -11 dB
strong interference, very unstable connection	-12 ... -15 dB
extreme interference, no connection possible	-16 ... -20 dB

2 Legal aspects

The following chapter addresses the legal difficulties that result from the operation of UAVs. First and foremost, legislation has to take safety aspects into consideration since crashing drones may cause damage to property of uninvolved people or - even worse - hurt them. Thus, UAVs need to be operated responsibly, which can be more or less insured by means of proper legislation. Mind that the following chapter is written in February of 2021 and regulations in this relatively new field may change rapidly.
The original plan was to introduce new regulations in July of 2020, but the implementation was postponed to January of 2021. The section *Former situation* describes the former situation while *New legal regulations* handles new laws that are enacted in 2021. Contrary to the former regulations from 2014, these will apply throughout the entire EU and not just Austria, which is a significant advantage for consumers and businesses as they will no longer need to inquire about legislation in every EU country they operate in.

2.1 Former situation

As already mentioned, UAV laws used to be determined by the individual member states of the EU. This section describes the former situation in Austria, determined by the *Luftfahrtgesetz* or *LFG* from 2014 [6]. This law divided UAVs without cameras for non-commercial use into three categories. The first category was split into three sub-categories for usage on model-plane airfields:

***Klasse Spielzeug* (Toy class)**

A UAV was considered to be a toy if its maximum kinetic energy was lower than 79 Joule. The relevant formula in this context was the well known relation

$$E = \frac{m \cdot v^2}{2} \tag{5}$$

with m being the UAV's mass and v being the maximum velocity that it can reach.

***Klasse Flugmodell* (Plane model class)**

This class contains non-commercial UAVs without cameras with a weight up to 25kg, which are operated within a range of 500m around a designated airfield. People or things must not be endangered and a line of sight connection must be guaranteed. This class needs no approval.

***Klasse Flugmodell über 25kg* (Plane model above 25kg class)**

This class contains non-commercial UAVs without cameras with a weight between 25 and 150 kg. The rules regarding maximum operating distance and LOS connection are the same as for UAVs up to 25kg. This class needs an approval.

As soon as the UAV carried a camera or was intended to be used commercially, it was categorized as a drone, see LFG §24 f and g. Drones were regulated, depending on the class that they belonged to. The two possible classes were *Klasse 1 uLFZ* (legislation category 2) and *Klasse 2 uLFZ* (legislation category 3), with *uLFZ* essentially meaning UAV. Whether a UAV could be operated legally was determined by it's class and the population density in the area of operation.
Category 2 included UAVs with a maximum take off weight (MTOW) between 5 and 150kg, which could

be operated comercially and further away than 500m from a designated airfield as long as visual line of sight (VLOS) conditions were guaranteed.
Every flight that exceeds these limits was included in category 3, to which rules of manned aviation were applied.

2.2 New legal regulations

In 2021, unified EU-wide rules were implemented [7]. Basically, UAVs (or rather, UAV flights) are now subdivided into two categories, **open** and **specific**. For high-risk flights, classical aviation rules must be obeyed. This extra category is called **certified**.

Open category

For open category flights, no takeoff approval is needed and the flights are subdivided into subcategories A1, A2 and A3. In most cases, an operator registration is required though.
The main difference between subcategories is the maximum takeoff mass or MTOM, while the main commonality of open flights is that they must be conducted in visual line of sight and in altitudes less than 120 meters. For more deatils, see figure 7.

<table>
<tr><th colspan="2">UAS</th><th colspan="3">Operation</th><th colspan="2">Operator/pilot</th></tr>
<tr><th>Class</th><th>MTOM</th><th>Subcategory</th><th>Operational restrictions</th><th>Distance from people</th><th>Operator Registration Required</th><th>Remote pilot competence</th></tr>
<tr><td>Privately built</td><td rowspan="2">< 250 g</td><td rowspan="2">A1</td><td rowspan="5">▪Operate in visual line of sight below 120 m altitude
▪Fly away from airports
▪Respect specific rules defined by the zone in which you operate</td><td rowspan="2">You can fly over uninvolved people (not over crowds)</td><td>No</td><td>Read owner manual</td></tr>
<tr><td>C0</td><td rowspan="3">Yes</td><td rowspan="2">▪Read owner manual
▪Perform online training
▪Pass online test</td></tr>
<tr><td>C1</td><td>< 900 g</td></tr>
<tr><td>C2</td><td>< 4 kg</td><td>A2</td><td>You can fly at a safe distance from uninvolved people</td><td>▪Read owner manual
▪Perform online training
▪Pass online test
▪Pass a theoretical test in a centre recognised by the aviation authority (only if you intend to fly close to non involved people)</td></tr>
<tr><td>C3</td><td rowspan="3">< 25 kg</td><td rowspan="3">A3</td><td rowspan="3">You should:
▪ Fly in an area where it is reasonably expected that no uninvolved people will be endangered
▪ Keep a safety distance from urban areas</td><td rowspan="3">▪ Read owner manual
▪ Perform online training
▪ Pass an online test</td></tr>
<tr><td>C4 (model aircraft)</td></tr>
<tr><td>Privately built</td></tr>
</table>

Figure 7: Comparison of the open flight category [7]

Specific category [8]

Should a flight not fulfill the restrictions of the open category, it falls under the specific category. In this case, national authorities (Austro Control in Austria) must give their approval.
For operators, the first step in this case should be to see if the operation falls under a standard scenario, which enables a quick approval. Should this not be the case, the standardized EASA risk assessment,

called Specific Operational Risk Assessment (SORA), needs to be conducted. If many similar SORAs are approved by the authorities, they may be used to form a standard scenario. As of today however, none of these exist and usually, a lengthy risk assessment is necessary.
This category is relevant in the context of this thesis as it regulates the UAV used for measurement flights.

3 Existing models for large scale fading and shadowing

One of the main goals of this thesis is to compare the models that result from the machine learning approach to existing ones. Unfortunately, most previous research is focused on large scale fading within buildings or urban areas. Since the test flights had to be conducted in rural areas, those models are pretty much useless for comparison with the gathered flight data. One exception is the Stanford university interim (SUI) [10] model, which can be used to model reception in rural areas with different degrees of vegetation.
This chapter offers a short explanation of fading and shadowing, followed by a comparison between the models found in the thesis and the SUI model for large scale fading.

3.1 Theoretical introduction

This section serves to give a quick recap of the theory behind fading and shadowing. Furthermore, the boundaries of what is investigated are defined: The goal of this thesis is to find a model for reception as a function of the distance, such that the effect of **large scale fading** is the one that emphasis is put on.
The models explained in the following sections are only valid in far field. Since all measurements were conducted in distances orders of magnitude larger than *lambda*, they can be assumed to be valid in the context of this thesis.

3.1.1 Small scale fading

The first fading phenomenon that is tackled is the so called small scale fading, which describes fluctuations in the received signal due to the arrival of multiple multipath components of the transmitted signal at the transmitter. These will be slightly delayed in time and thus also be received at a different phase angle. Furthermore, electromagnetic waves experience a phase jump of 180 degrees when being reflected by a conductor, which leads to effects on the received power due to destructive or constructive interference. Overall, there are three main effects of small scale fading (Rappaport, P.139):

Rapid changes in received power over distances in the order of magnitude of λ or over short time periods due to destructive or constructive interference of signals arriving at the receiver.

Random frequency modulation since multipath components arrive from different angles, which means that they experience different Doppler shifts if the receiver is moving.

Time dispersion due to the different ways covered by multipath components (echoes), which result in different propagation delays of signals overlapping at the receiver.

It is important to note that the first and the third effect even occur if the receiver is stationary since surrounding objects may move, e.g. cars in a city. If the received fields combine in such a way that the received power is close to zero, this is called a *deep null*. In reality, such a condition usually only persists for a short period of time since either the mobile or objects that scatter are moving. However, if deep nulls are a problem, this can be solved by using spatially diverse antennas and cross-polarization.
Since one channel (at a fixed point in time) has different small scale fading properties for different wavelengths (i.e. frequencies), the phenomenon can be described by means of a transfer function. This way

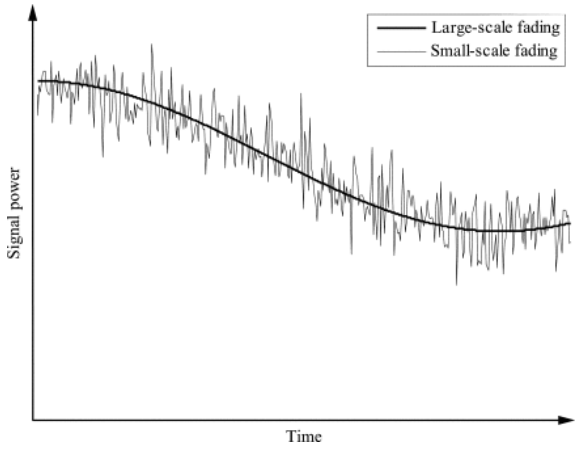


Figure 8: Illustration of temporal large- and small-scale fading [13], analogous to spatial fading

of thinking of a channel becomes more relevant for signals with higher bandwidth since their frequency components will be affected in a different way. The so called *channel bandwidth* or coherence bandwidth describes a frequency range in which the channel has a transfer function that stays within certain boundaries. This does not mean that the channel has good transfer characteristics (i.e. low attenuation) within this range. The transfer function only needs to be within said boundaries, since the transmitted signal would otherwise be distorted.
This effect can be mitigated by frequency division multiplexing. This means that the transmission is divided into multiple channels with bandwidths smaller than the coherence bandwidth, such that every subchannel suffers from little distortion. Usually, such systems use error correcting codes to handle those subchannels which fall onto deep nulls.

3.1.2 Large scale fading and shadowing

Other than the previously discussed small scale fading, large scale fading describes the macroscopic behavior of reception quality, usually over a range of hundreds of meters in case of mobile communication systems. The usual approach to retrieve a model for this kind of fading is to conduct multiple measurements at different points in space/time and average them to get rid of small scaling effects. For spatial measurements, data is averaged over a so called measurement track, which usually ranges from 5λ to 40λ (Rappaport, P.70). These averaged measurements are then plotted versus the distance, often on a log-log or semilogarithmic scale.
In free space (no obstructions or other factors, which increase attenuation, like raindrops), large scale fading occurs due to the increasing surface of a given spatial angle for increasing radii. This fact is taken into account by the well known Friis free space equation

$$P_r(d) = \frac{P_t G_t G_r \lambda^2}{(4\pi)^2 d^2 L} \tag{6}$$

where d is the distance between receiver and transmitter. P_r, G_r, P_t and G_t are the receiver and transmitter power and gain, respectively. The parameter L takes miscellaneous losses into account. The denominator

contains the expression for a spherical surface with radius d: $A = 4\pi d^2$

It is useful to use double logarithmic plots when depicting path loss since the relation between d and P_r ($P_r \sim d^{-2}$) then leads to a linear relationship with slope -2 on the plot. Furthermore, variations in parameters simply lead to an offset on this kind of plot, see figures 9(a) and 9(b). Doubling a parameter leads to a constant offset of 3dB, if it is multiplied by ten, the gain/attenuation is 10dB.

The slope of this log-log plot is called the path loss exponent n. It can be used to model different propagation conditions like urban areas or a multi-storey office building. The following table shows some values of n for different surroundings (Rappaport, P.104):

Environment	Path loss exponent n
Free Space	2
Urban area cellular radio	2.7 to 3.5
Shadowed urban cellular radio	3 to 5
In building line-of-sight	1.6 to 1.8
Obstructed in building	4 to 6
Obstructed in factories	2 to 3

Another common way to describe these relationships is via the average path loss equation. It describes the average attenuation that the signal experiences on it way from receiver to transmitter as a function of the distance (Rappaport, P.102):

$$\overline{PL}(d)[dB] = \overline{PL}(d_0) + 10n \cdot log_{10}\left(\frac{d}{d_0}\right) \tag{7}$$

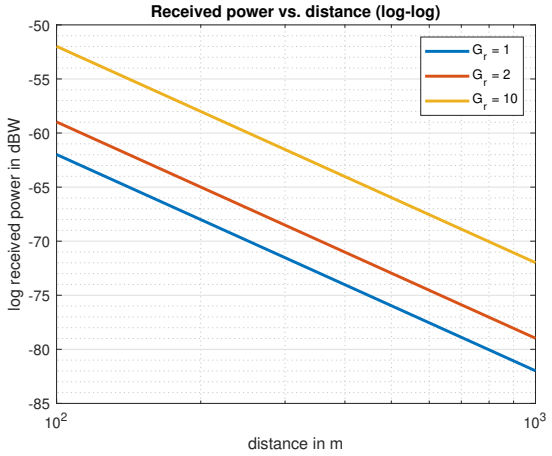
It it obvious that this model only describes the average behavior. In reality, different receivers will experience different reception qualities due to different degrees of cluttering, even if they are at the same distance from the transmitter (see figure 10). This behavior can be modeled by so called log-normal shadowing:

The path loss at a given point in space is random and normally distributed around the average path loss for the point's radius on the log-log plot. Shadowing is thus modeled by a normal distribution on the logarithmic plot (i.e. a normal distribution in dB).

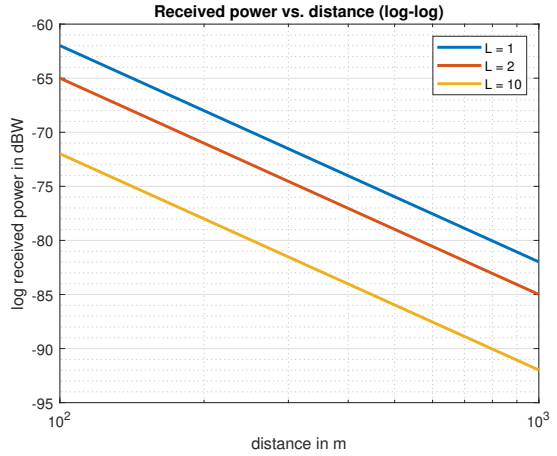
$$PL(d)[dB] = \overline{PL}(d) + X_\sigma = \overline{PL}(d_0) + 10n \cdot log_{10}\left(\frac{d}{d_0}\right) + X_\sigma \tag{8}$$

X_σ (in dB!) is a zero-mean, normally distributed random variable with variance σ (also in dB).

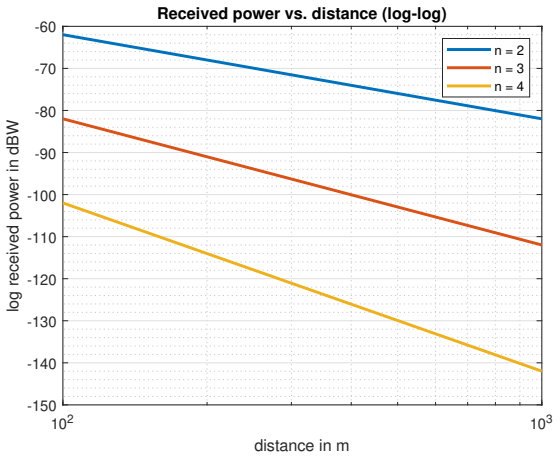
Finally, it is to be mentioned that shadowing does not play a major in the context of this thesis, since the focus is set on large scale fading and measurements are conducted in a rural are. Still, it is important to know the basics of shadowing (and also small scale fading!) when dealing with reception quality data.



(a) Log received power with variable receiver gains



(b) Log received power with variable loss coefficients



(c) Log received power with variable path loss exponents

Figure 9: Variations in parameters are easily depicted on a logarithmic scale

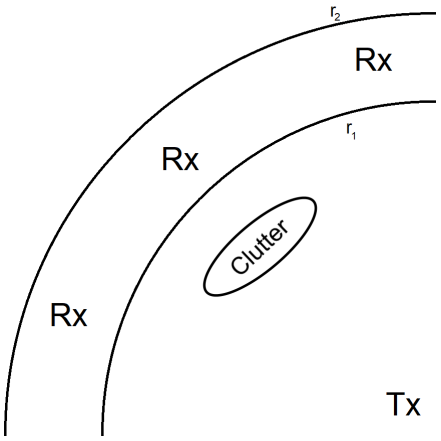


Figure 10: The receiver behind the clutter will probably suffer from worse reception quality, even though the radius is the same. This is not taken into account in the simple path loss equation (see equation 7).

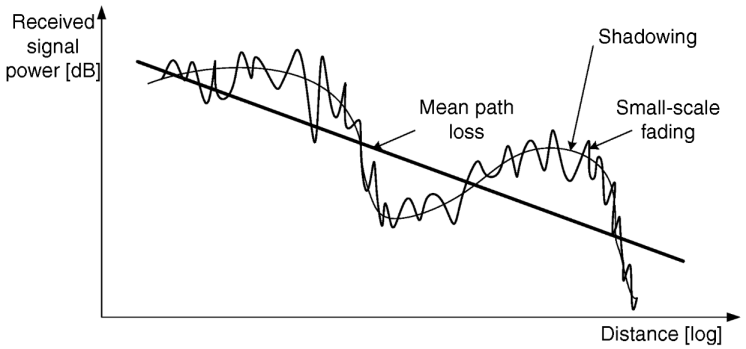


Figure 11: Illustration of spatial fading and shadowing [14]

3.2 The Stanford University Interim (SUI) model

As already mentioned previously, the SUI model [10] is the one that the results of this thesis are compared to. It can be used to model reception in rural (or suburban) areas with different degrees of vegetation or mountainous terrain. **The receiver antenna has to be below 2m, according to the model's definition. This condition is not fulfilled in this thesis!**

A model with correction factors has been published, which is intended to be used for receiver heights up to ten meters, which is of course still lower than typical UAV operation altitudes. Regardless, it is the model that best suits the needs as there are no existing models for path loss in this frequency range that can be used for the altitudes that UAVs operate in.

There are extensions to this model, which are used for urban areas, but for this thesis, the SUI model (also called Erceg model) with correction factors is used. The basic path loss formula (without correction factors for higher altitudes) is similar to equation 7:

$$PL = A + 10\gamma \cdot log_{10}\left(\frac{d}{d_0}\right) + s \tag{9}$$

where

$$d > d_0$$

$$A = 20 \cdot log_{10}\left(\frac{4\pi d_0}{\lambda}\right)$$

$$\gamma = a - b \cdot h_b + \frac{c}{h_b}$$

$$d_0 = 100m$$

$$10m < h_b < 80m$$

$$8.2dB < s < 10.6dB$$

It takes the distance d, the wavelength λ, the base station height h_b, the shadowing effect s and the terrain parameters a, b and c into account. Since the UAV operates at a significant distance from ground, it is safe to assume that $s = 8.2dB$.
As already mentioned, the model is suited to different degrees of vegetation. There are three categories:

Category A

Hilly terrain with moderate-to-heavy tree densities, which results in the maximum path loss.

Category B

Hilly environment but rare vegetation, or high vegetation but flat terrain. Intermediate path loss condition is typical of this category.

Category C

Mostly flat terrain with light tree densities. It corresponds to minimum path loss conditions.

The terrain that was used to conduct the test is quite flat, which justifies the use of category C. Furthermore, it is safe to assume that the high operating altitudes lead to a more "Category-C-ish" behavior of the reception quality. The following table shows the parameters a, b and c for the different types of terrain:

	Category A	Category B	Category C
a	4.6	4	3.6
b	0.0075	0.0065	0.005
c	12.6	17.1	20

3.3 The SUI model with correction factors

In this subsection, a correction factor for higher receiver antenna heights is introduced. This factor leads to an additional additive term in the logarithmic expression for the path loss:

$$PL = A + 10\gamma \cdot log_{10}\left(\frac{d}{d_0}\right) + s + \Delta PL_h \tag{10}$$

with

$$\Delta PL_h = \begin{cases} -10.8log(\frac{h}{2}) & \text{for terrain type A and [sic!]} \\ -20log(\frac{h}{2}) & \text{for terrain type C} \end{cases} \tag{11}$$

It can be assumed that the upper case in equation 11 is valid for terrain type A and B, but the line can be found at [2] as cited above. Since the terrain type at the test site is of category C anyway, the path loss equation **for receiver heights up to ten meters** can be written as:

$$PL = A + 10\gamma \cdot log_{10}\left(\frac{d}{d_0}\right) + s - 20log(\frac{h}{2}) \tag{12}$$

It has to be tested whether this model is applicable in the given scenario and which adaptations will need to be conducted.

Basically, this model predicts a lower path loss that decreases with altitude. This makes sense near ground as less cluttering can be expected for higher receiver altitudes. In case of a UAV flight however, this effect can be assumed to be non-existent as there are no objects (except for maybe birds) that badly influence reception in such high altitudes.

3.4 Proposed models

In the following section, the large scale fading models proposed by the author are presented. They were thought up after a qualitative and quantitative analysis of reception parameter behavior.

An actual data fitting was conducted only for the path loss model. The proposed model for reception parameters can rather be seen as a corollary, serving as a starting point for further investigation.

3.4.1 Proposed model for reception parameters

Since the factors that usually contribute to lower reception quality (like non-line of sight connection, dense population of urban areas etc.) don't play a role for the recorded flights (they were conducted in rural areas and in high altitudes), it makes sense to look at another factor that plays a major role: The elevation angle between UAV and base station.

Since the LTE network is designed to provide internet connection in ground proximity, antennas are designed accordingly and direct as little power as possible towards higher altitudes, i.e. they have a very pronounced main lobe around $\theta = 0°$. Antenna patterns can be found at [11], pages 7 and 8.

This factor makes it useful to include the elevation angle θ into the model. Of course, reception is also possible in higher altitudes due to signals being reflected from the ground, but on average, reception parameters

will exhibit lower values in these cases, see figures 12 and 13.

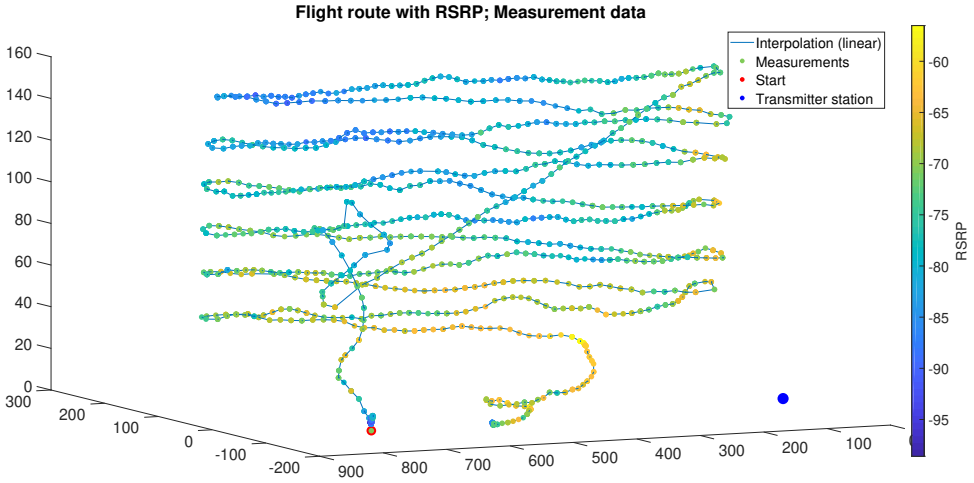


Figure 12: One of the flight routes with the respective RSRP values

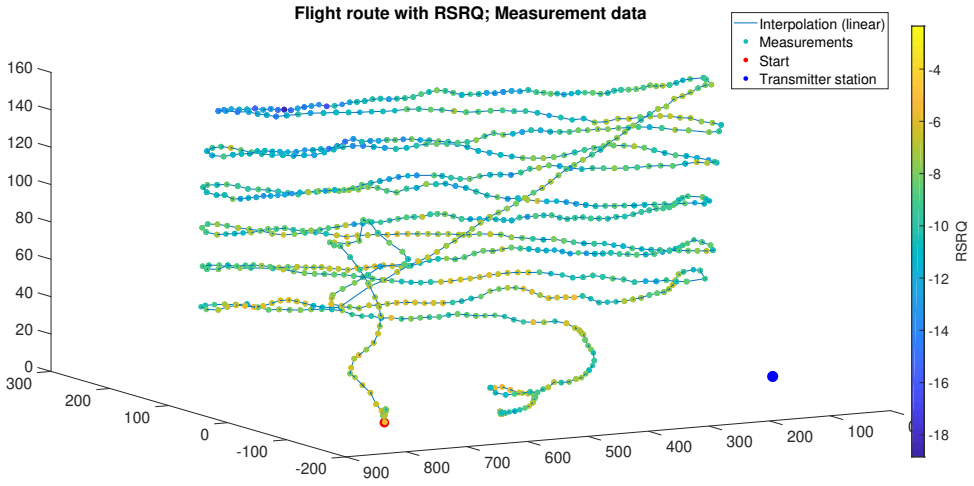


Figure 13: One of the flight routes with the respective RSRQ values

Since RSRP measures the total power on certain sub-carriers, it is impacted stronger by higher altitudes. This is due to the fact that other devices that emit electromagnetic waves in the same frequency region have a stronger impact closer to the ground. However, the RSRQ value is also impacted significantly by higher elevation angles, which leads to the conclusion that way less useful signal is picked up in these heights.
Due to these factors, it makes sense to look for a reception parameter model that includes the elevation angle and not just the transmit power and the path loss, see equation 13 below. $f_{RSRP}(\theta)$ is a function that should fit the measurement data as well as possible. It is greatly influenced by the antenna pattern, but also by the type of environment as reflected signals also contribute to reception quality.

$$RSRP(d,\theta) = RSRP(d_0) - c_{RSRP} \cdot log_{10}\left(\frac{d}{d_0}\right) - f_{RSRP}(\theta) \tag{13}$$

For the other parameters, analogous formulas can be found. The example in equation 14 below shows the formula for RSRQ, which will need a different function $f_{RSRQ}(\theta)$ to take the elevation into account.

$$RSRQ(d,\theta) = RSRQ(d_0) - c_{RSRQ} \cdot log_{10}\left(\frac{d}{d_0}\right) - f_{RSRQ}(\theta) \tag{14}$$

The parameters c_{RSRP} and c_{RSRQ} determine the slope of the path loss on the log-log plot.
Even though the dependence on θ is clearly visible on figures 12 and 12, an anomaly can be observed for very large elevation angles. This might be due to a reflecting surface like a car, that coincidentally reflected the antenna signal in this manner.

3.4.2 Proposed model for reception path loss

Since the elevation angle is kept constant for path loss calculations, the path loss model does not take the elevation into account. These considerations are a relatively new field, such that a valid starting point is a model where $PL(d_0)$ and $PL\left(log_{10}\left(\frac{d}{d_0}\right)\right)$ is extracted from the given measurements and not predicted by factors like antenna height etc.
Since the measurements were conducted in rural areas with little interference, the evolution of RSRP as a function of the log distance serves as an approximation for the evolution of the path loss:

$$\frac{dPL(r)}{dr} \approx \frac{dRSRP(r)}{dr} \tag{15}$$

The proposal in this thesis is a path loss model that follows the typical linear equation scheme

$$y = k \cdot x + d \tag{16}$$

with d representing the path loss at distance d_0, i.e. $PL(d_0)$ and $k \cdot x$ representing the sloped line on the log-log plot, i.e. $c_{PL} \cdot log_{10}\left(\frac{d}{d_0}\right)$.
Thus, the following model is proposed:

$$P(d)|_{dBW} = \underbrace{P(d_0)_{dBW} - PL_0|_{dB}}_{PL(d_0)->d} - \underbrace{c_{PL} \cdot log_{10}\left(\frac{d}{d_0}\right)}_{PL(d)->k\cdot x} \tag{17}$$

With $PL(d_0)$ determined by means of measurements, the constant term $PL_0|_{dB}$ can be calculated if the log transmit power $P(d_0)_{dBW}$ is known. A prediction of $PL_0|_{dB}$ is not yet possible as more measurements need to be conducted.

3.5 Model comparison

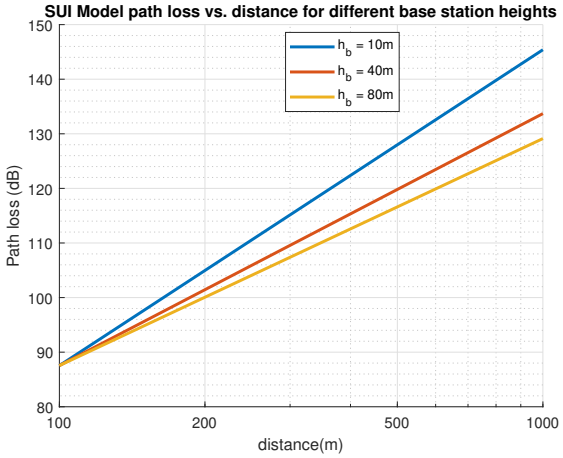
This section serves as a comparison between the mentioned models, detailing their differences.

3.5.1 The SUI model

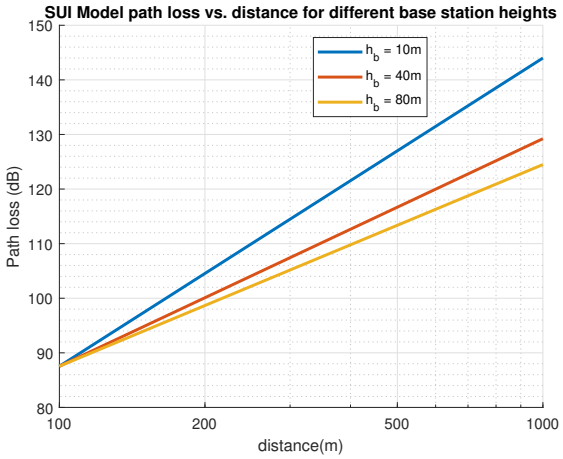
For the basic SUI path loss model (see equation 9), there are basically two parameters: s and γ. The first one gets added to the logarithmic equation, simply adding an offset on the logarithmic plot, corresponding to a factor in actual path loss.
The second parameter γ takes the surrounding terrain and the base station height h_b into account and leads to different slopes on the log-log plot. For results, see figure 14.

3.5.2 The SUI model with correction factors

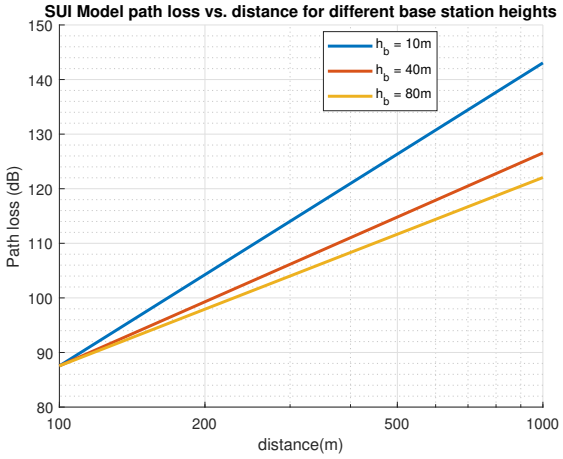
This extension simply adds a factor to the path loss equation for higher receiver altitudes, see equations 10 and 11. This correction factor leads to an offset in the path loss. Higher receiver antennas usually lead to better reception as there is usually less cluttering in higher altitudes. A comparison for different antenna heights can be found in figure 15. All comparisons were conducted with a shadowing parameter of $s = 10dB$.
Figure 16 depicts the correction offset as a function of receiver antenna height. As one would expect, the effect is most pronounced for hardly populated terrain since trees and buildings may very well be higher than ten meters, leading to shadowing, even for higher receiver antennas.



(a) Terrain category A

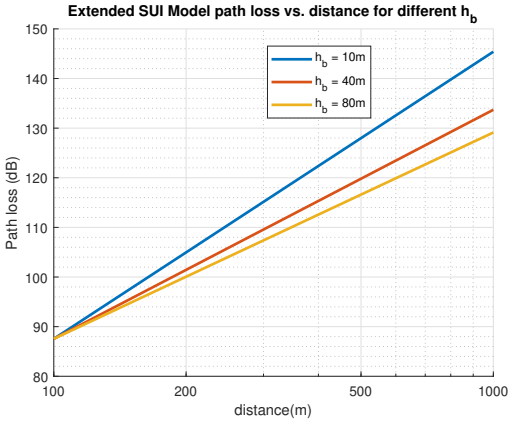


(b) Terrain category B

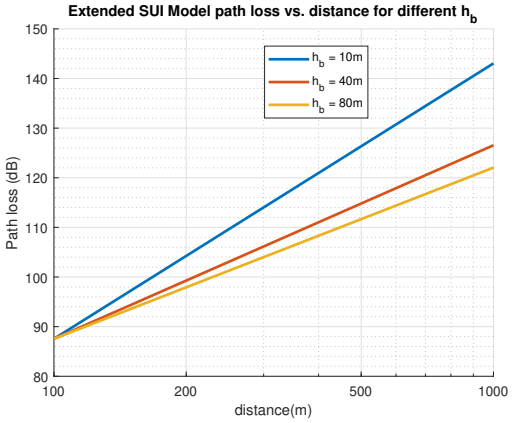


(c) Terrain category C

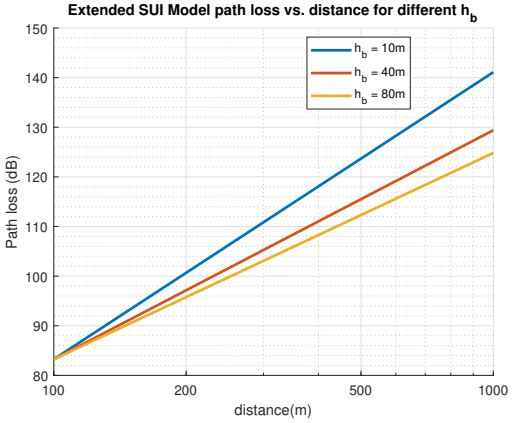
Figure 14: The SUI model predicts different path loss lopes for different kind of terrain categories and for different base station antenna height. The second parameters s (shadowing effect parameter) is a constant offset on the log-log plot ($s = 10dB$ in this plot).



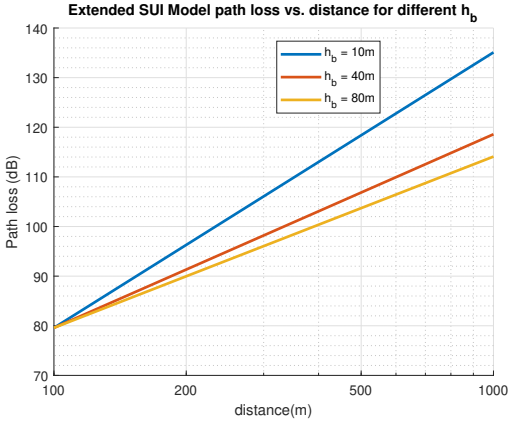
(a) Receiver antenna height of 2m, Terrain category A



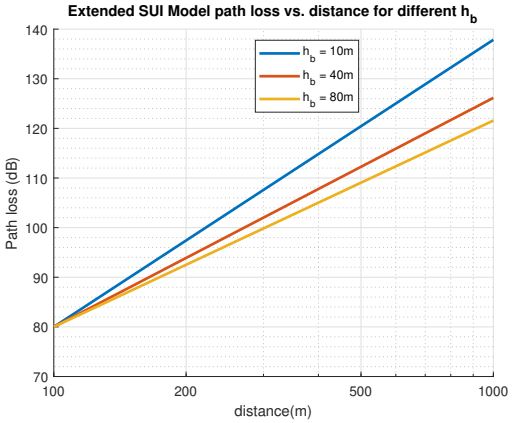
(b) Receiver antenna height of 2m, Terrain category C



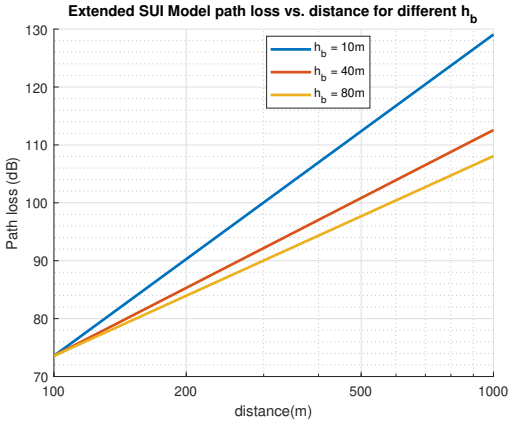
(c) Receiver antenna height of 5m, Terrain category A



(d) Receiver antenna height of 5m, Terrain category C



(e) Receiver antenna height of 10m, Terrain category C



(f) Receiver antenna height of 10m, Terrain category C

Figure 15: Higher receiver antennas lead to less path loss. The comparison is done only between category A and C since cat B behaves similarly to cat A.

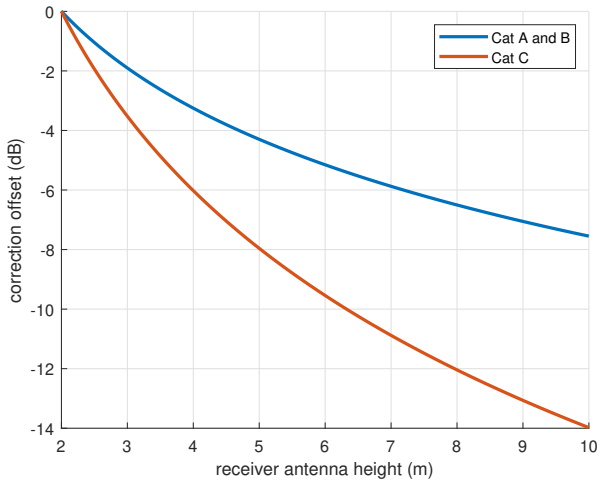


Figure 16: The correction factor/offset depends on the kind of terrain and on the receiver antenna height.

3.5.3 Proposed model for reception parameters

Since the used base station antennas have a strongly pronounced main lobe and hardly direct energy to higher altitudes [11], a non line of sight (NLOS) connection is to be expected for elevation angles higher than a certain threshold. f_{RSRP} and f_{RSRQ} in equations 13 and 14 must be chosen accordingly, for example as a roll off cosine or a step function.

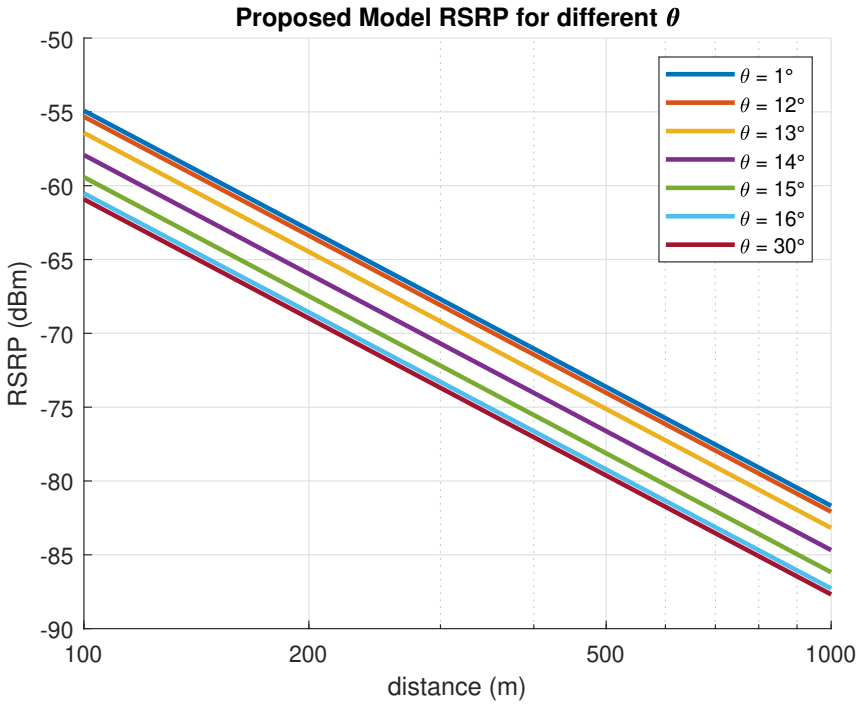
Roll off cosine

For a continuous transition from low-theta to high-theta reception conditions, a roll off cosine can be used. The proposed model would then exhibit the form

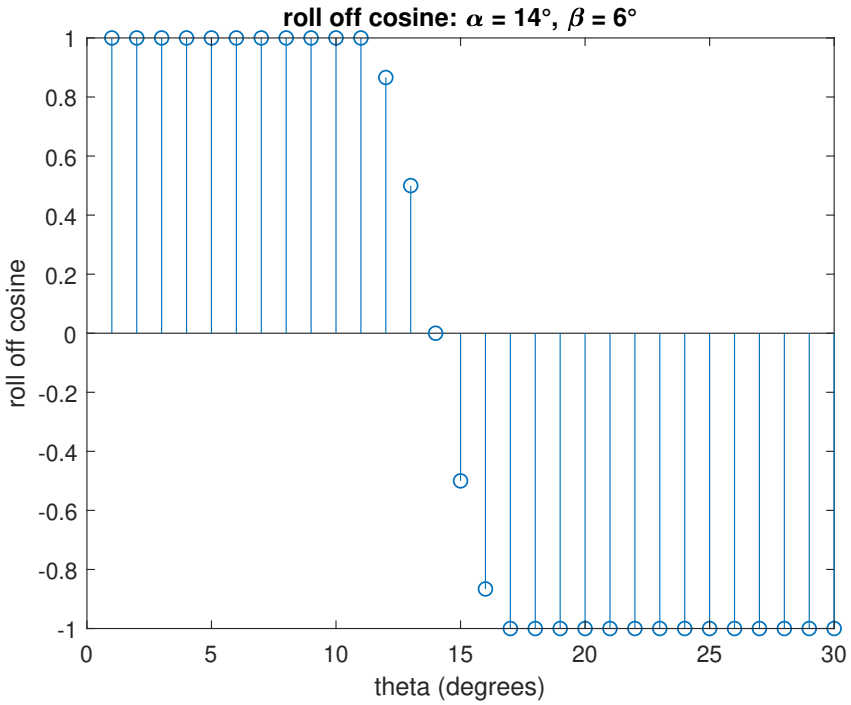
$$RSRP(d,\theta) = \begin{cases} RSRP(d_0) + c_{RSRP} \cdot log_{10}\left(\frac{d}{d_0}\right) + A & \text{if } \theta < \alpha - \frac{\beta}{2} \\ RSRP(d_0) + c_{RSRP} \cdot log_{10}\left(\frac{d}{d_0}\right) + A \cdot cos\left(\frac{\theta - \alpha + \beta/2}{\beta} \cdot \pi\right) & \text{if } \alpha - \frac{\beta}{2} < \theta < \alpha + \frac{\beta}{2} \\ RSRP(d_0) + c_{RSRP} \cdot log_{10}\left(\frac{d}{d_0}\right) - A & \text{if } \theta > \alpha + \frac{\beta}{2} \end{cases} \quad (18)$$

With α being the threshold and β being the transition width. A describes the amplitude, i.e. the impact of the elevation angle. Using these two parameters, different transitions (smooth vs. abrupt, low vs. high transition amplitude) can be modeled.

An analogous model can be used for RSRQ.

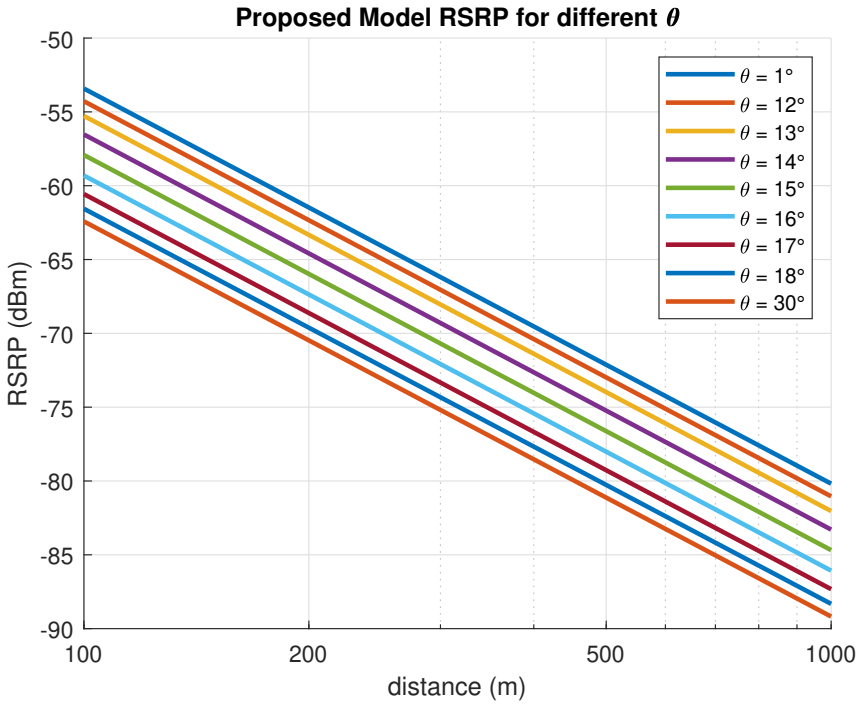


(a) RSRP over distance for different elevation angles, Rcos model

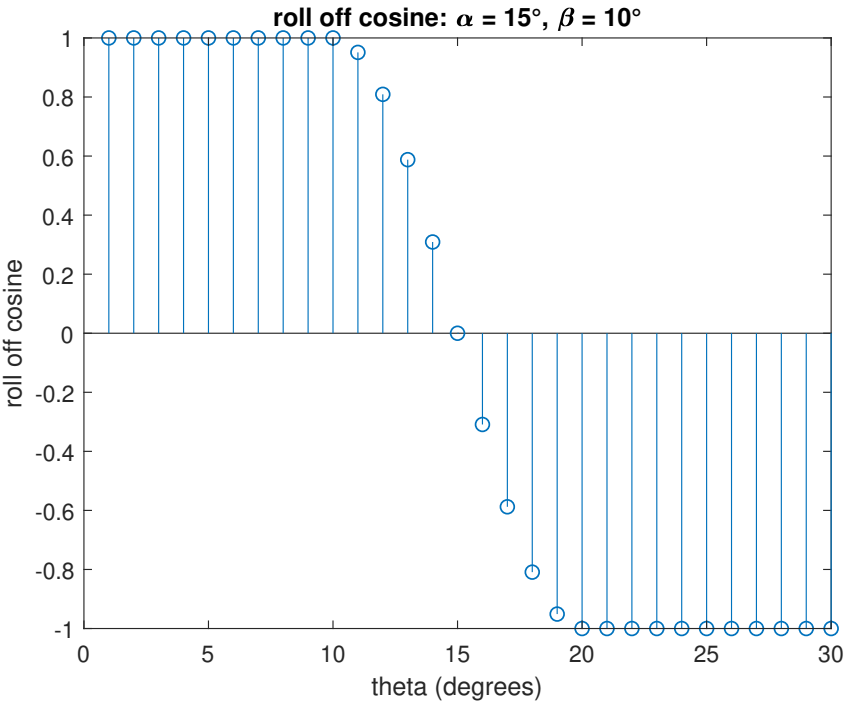


(b) The corresponding roll off cosine

Figure 17: Possible predictions of the roll off cosine model with parameters $\alpha = 14°$ and $\beta = 6°$. The transition amplitude is set to 6dB.



(a) RSRP over distance for different elevation angles, Rcos model



(b) The corresponding roll off cosine

Figure 18: Possible predictions of the roll off cosine model with parameters $\alpha = 15°$ and $\beta = 10°$. The transition amplitude is set to 9dB.

Step function

Due to strong directional base station antenna characteristics, a step function should be considered. This function is also useful to choose between two different control algorithms if the UAV is operating autonomously since it only puts out two values.

$$RSRP(d,\theta) = \begin{cases} RSRP(d_0) + c_{RSRP} \cdot log_{10}\left(\frac{d}{d_0}\right) + A & \text{if } \theta < \alpha \\ RSRP(d_0) + c_{RSRP} \cdot log_{10}\left(\frac{d}{d_0}\right) - A & \text{if } \theta > \alpha \end{cases} \tag{19}$$

Same as before, an analogous equation can be used to model RSRQ.

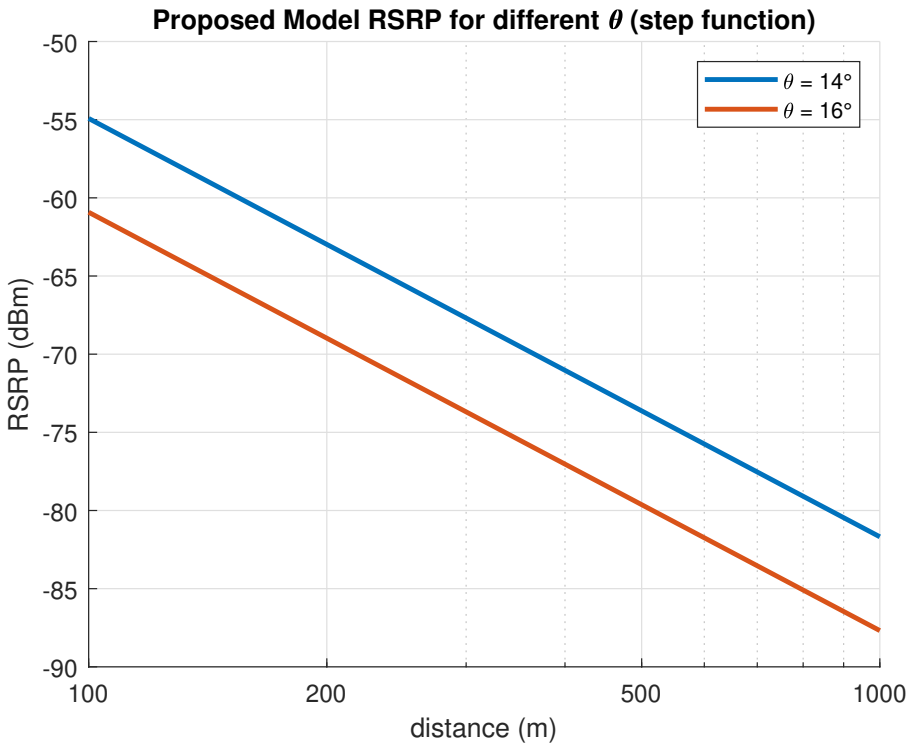


Figure 19: Possible relation between RSRP and d for different elevation angles. The transition is very sharp because a step function was used to model it.

3.5.4 Proposed model for path loss

Other than reception parameters, path loss per definition describes the loss of received power as a function of the distance. Thus, azimuth and elevation angles are kept constant (or rather, within defined constraints) and the only argument for the path loss function is the distance between receiver and transmitter:

$$P(d)|_{dBW} = \underbrace{P(d_0)_{dBW} - PL_0|_{dB}}_{PL(d_0)->d} - \underbrace{c_{PL} \cdot log_{10}\left(\frac{d}{d_0}\right)}_{PL(d)->k \cdot x} \tag{20}$$

As already discussed, this equation leads to a linear function $y = k \cdot x + d$ on a log-log plot. In order to determine the offset and slope, the azimuth angle constraints were chosen such that only the "wall" that was flown is selected. Then, the elevation angle was looped through in one degree steps, calculating slope and offset in every iteration. The end result consists of the average values for k and d, weighted by the amount of data points in every "corridor". These ideas are depicted in figures 20 and 21.
As already discussed, the path loss estimation is based on the RSRP. Furthermore, directional characteristics of the receiver antenna were not taken into account. Thus, there measurements are not exact, but still lead to plausible results. Furthermore, reception quality parameters were always in regions where no signal interruptions are to be expected. This hints at safe operation, at least in the used frequency, which is at 800MHz. For higher frequencies, a more pronounced path loss is to be expected.
In figure 23, it becomes apparent that lower elevation angles tend to show a steeper decrease in RSRP. Thus, even though the weighted mean predicts a proper signal reception for a distance up to 3 km, a margin of safety should be used.
As a conclusion, it must be mentioned that the model at hand is not as elaborate as other models (like the SUI model), but since the modeling of reception parameters in such altitudes is a new field, it serves as a starting point for others, who further refine it.

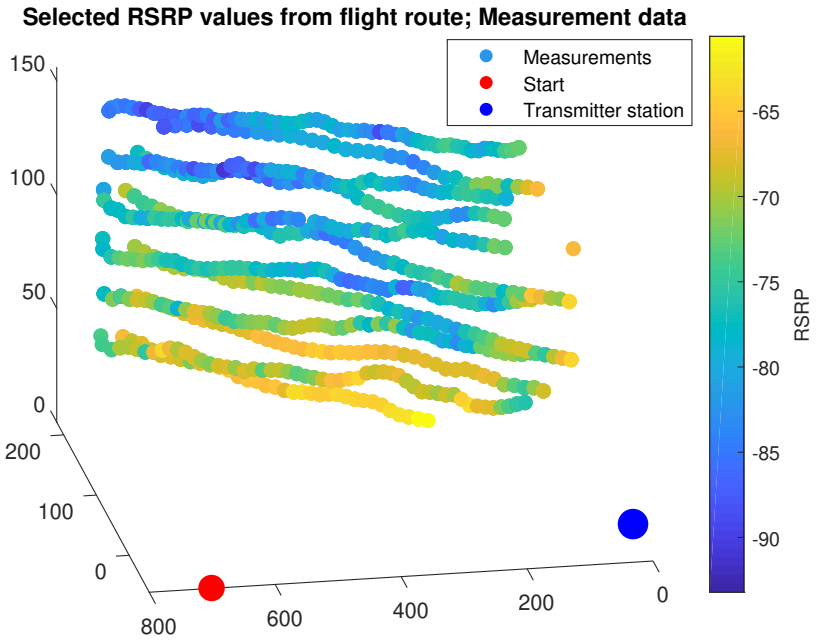


Figure 20: One of the so called "walls". The azimuth angle is restricted to a 3 degree interval, such that the UAV's way from the start (red point) and back is filtered out.

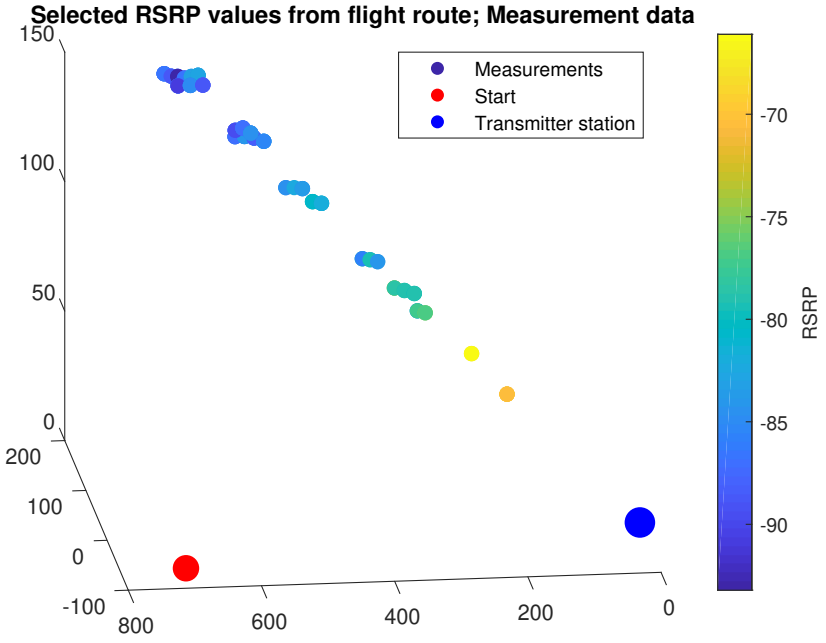


Figure 21: One of the so called "corridors". Starting from the previously filtered wall, the elevation angle was confined to a one degree interval.

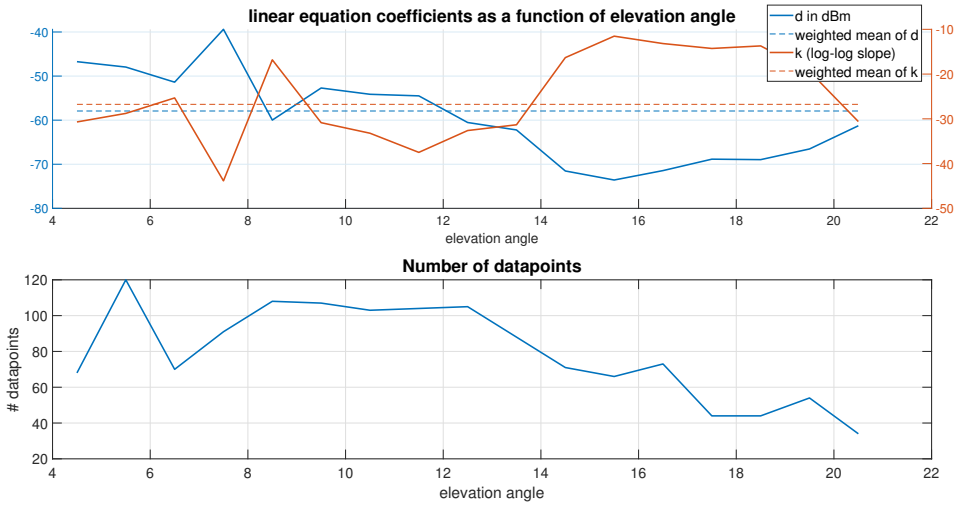


Figure 22: Different corridors lead to different slopes and offsets. Data with more data points is more reliable and more significant. Thus, the weighted average is depicted as dashed line on the upper plot.

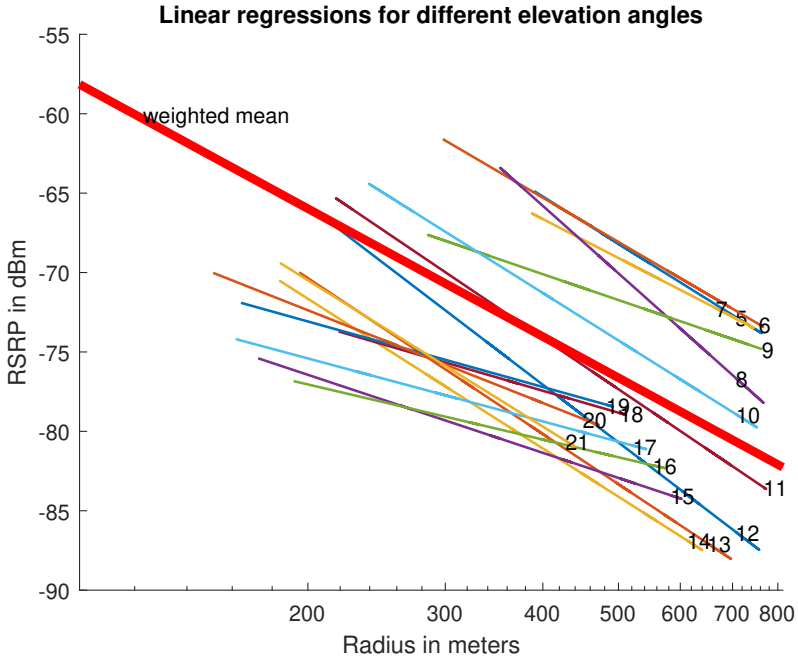


Figure 23: All corridor regressions for one of the walls and their weighted mean. If the assumption that a RSRP of -100dB suffices is made, the range would be larger than 3 kilometers.

4 Theory of used Technologies

The following chapter contains an overview of the necessary theoretical backgrounds. For both technologies, the focus will be set on the aspects that are actually used in this work, i.e.:

The theory behind linear regression will be described to the necessary extent. This means that only simple linear regression will be elaborated. Simple linear regression means that the relationship between *one* independent explanatory variable and one dependent response variable is described, as opposed to multivariate regression, which maps *multiple* explanatory variables to one response variable.
In this thesis, the relationship between distance (independent) and RSRP (dependent) is investigated, which means that multivariate regression will not be of interest.

Deep neural networks are structures with at least one hidden layer between input and output nodes. They are useful if a relationship between variables is too complex for standard linear regression and for a wide array of other tasks. For example, recursions within a deep neural network can be useful for time series prediction tasks as they add a kind of memory to the network, see figure 24.
Since the scope of this thesis is the modeling of a multivariate relationship, such structures will not be investigated. Instead, there will be a restriction to so called **feedforward neural networks**, which are structures that pipe the input data towards the output over the hidden layers without any recursions or other "unorthodox" methods.
In this thesis, they will be used to model the relationship between the UAV's position (Cartesian or polar coordinates, three input variables) and the RSRP/RSRQ values.

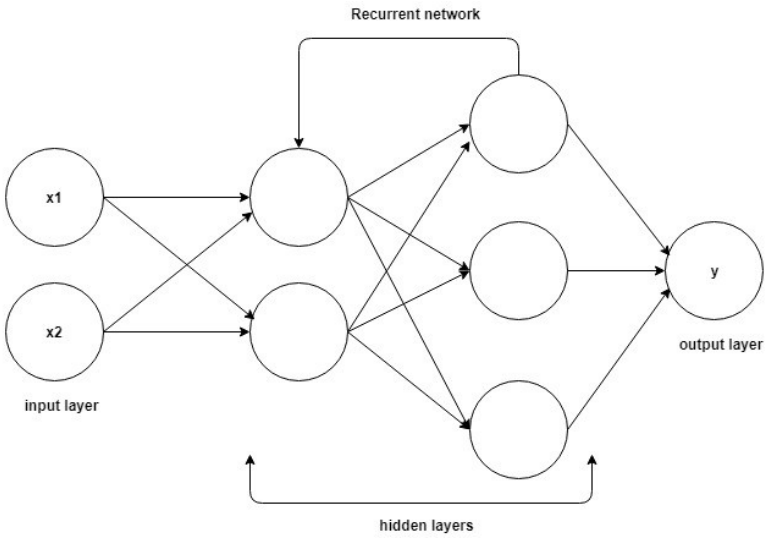


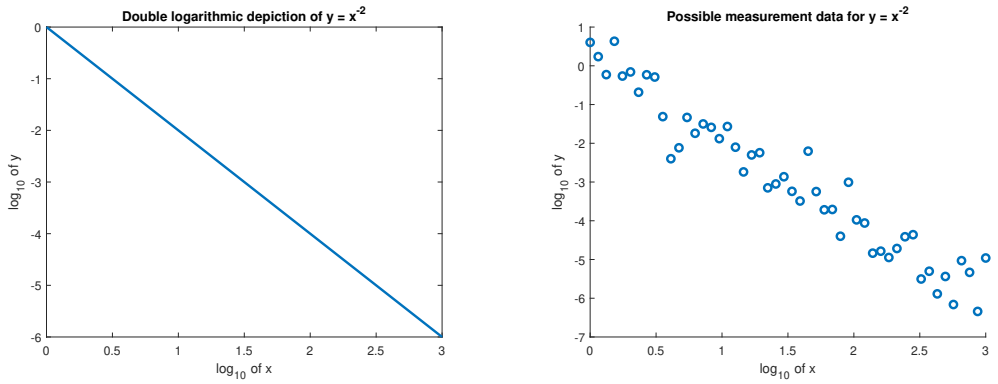
Figure 24: An example of a deep neural network with two hidden layers and a recurrent structure [15]

Both technologies stem from the field of machine learning. Linear regression is one of the more simple approaches, while deep neural networks are way more complex (and powerful).

4.1 Linear regression

In many (technical) applications, relationships between two variables can be described by means of a linear relationship. This is desirable since this is the simplest kind relationship that variables can exhibit and because the theory behind linear algebra is well understood and usually extensively taught in any engineering study program. Thus, it is often the case that nonlinear relations are approximated as linear ones in order to be better able to handle them.

One such example is electromagnetic wave propagation in the far field: Since the power diminishes with $\frac{1}{r^2}$, the relationship can be depicted as a linear function with a slope of -2 on a log-log-diagram.



(a) Ideal graph

(b) Possible measurement results

Figure 25: An ideal relationship versus possible measurements

Connecting these measurement dots with curved line segments would not make sense since it would not represent the actually occurring behavior. Instead, the aim of linear regression is to find a straight line that best fits the measurements.

This is done by means of linear algebra. The combination of measurements and their respective distances, i.e. the relation between the independent variable x and the response variable y can be written in matrix form:

$$\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon} \tag{21}$$

where

$$\boldsymbol{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_M \end{pmatrix} \tag{22}$$

denotes the measurement vector. It contains M elements.

$$\boldsymbol{X} = \begin{pmatrix} 1 & x_{11} \\ 1 & x_{21} \\ \vdots & \vdots \\ 1 & x_{M1} \end{pmatrix} \tag{23}$$

$\boldsymbol{X}$ contains the explanatory variables, i.e. the distances and a column of ones. This column serves serves to model the offset while the right column models the linear behavior. In the case of multivariate regression, this matrix would contain more columns to take multiple explanatory variables into account.

$$\boldsymbol{\beta} = \begin{pmatrix} \beta_0 \\ \beta_1 \end{pmatrix} \tag{24}$$

$\boldsymbol{\beta}$ is the so called parameter vector. Looking at equation 21, it becomes obvious that this vector is multiplied with $\boldsymbol{X}$. Thus, β_0 denotes the constant term, also known as the intercept, while β_1 denotes the slope. The goal of linear regression is to compute $\boldsymbol{\beta}$.
$\boldsymbol{\varepsilon}$ denotes the error term, which models the deviation of the measurements from the linear model.
In general, if a graph of order N is to be described, there are M = N+1 points necessary to do so. Since the performed regression is linear, N is equal to one. This leads to three possible scenarios:

M < N+1

The number of measurement points is too low. This means that the equation system is underdetermined and there is at least one degree of freedom. This case is irrelevant for linear regression.

M = N+1

The equation system is exactly determined, there are no degrees of freedom. In this case, $\boldsymbol{X}$ is a square matrix and thus invertible. The equation system can be solved by means of the following equation:

$$\boldsymbol{\beta} = \boldsymbol{X}^{-1}(\boldsymbol{y} - \boldsymbol{\varepsilon}) \tag{25}$$

This relatively trivial case does not require linear regression.

M > N+1

The number of measurement points is larger than the order of the used polynomial plus one, see figure 25(b). This means that $\boldsymbol{X}$ is not invertible. In this case, a way has to be found to approximate the relationship, taking into account all measurements.

The least squares cost function and the pseudoinverse

One very common way to choose the parameter vector $\boldsymbol{\beta}$ is to select the slope and the intercept such that the sum of the squared errors between the regression line and the measurements is minimized. The squared error is well suited as it results in positive values, independent of the sign of the individual error terms. Furthermore, it stronger penalizes higher error values because of the quadratic relationship.
The sum of the squared errors is the so called **least squares** cost function J_{LS}. It describes how bad the model works and thus needs to be minimized.

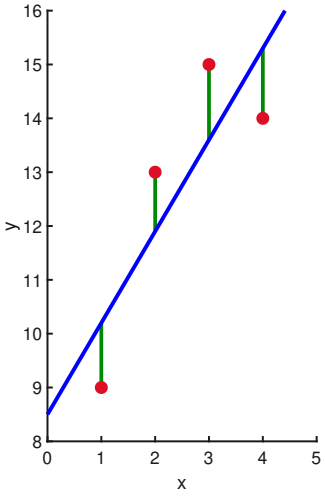


Figure 26: Illustration of the error term (green) in a linear regression

$$J_{LS} = \underbrace{||(\boldsymbol{y} - \boldsymbol{X\beta})||^2}_{\text{square norm}} = (\boldsymbol{y} - \boldsymbol{X\beta})^T(\boldsymbol{y} - \boldsymbol{X\beta}) \tag{26}$$

To do so, the gradient (vectorial derivative) with respect to $\boldsymbol{\beta}$ has to be taken and set to zero. Furthermore, some identities were applied to perform the multiplication of the braces in equation 26.

$$\nabla_{\boldsymbol{\beta}} J_{LS}(\boldsymbol{\beta}^T\boldsymbol{X}^T\boldsymbol{X\beta} - 2\boldsymbol{y}^T\boldsymbol{X\beta} + \boldsymbol{y}^T\boldsymbol{y}) = \boldsymbol{0} \tag{27}$$

Computing the gradient leads to:

$$2\boldsymbol{X}^T\boldsymbol{X\beta} - 2\boldsymbol{X}^T\boldsymbol{y} + \boldsymbol{0} = \boldsymbol{0} \tag{28}$$

$$2\boldsymbol{X}^T\boldsymbol{X\beta} = 2\boldsymbol{X}^T\boldsymbol{y} \tag{29}$$

In the next step, equation 29 is multiplied with $\frac{1}{2}(\boldsymbol{X}^T\boldsymbol{X})^{-1}$ from the left. This leads to the least squares solution $\boldsymbol{\beta}_{LS}$:

$$\boldsymbol{\beta}_{LS} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}\boldsymbol{y} \tag{30}$$

Equation 30 is the so called Moore Penrose pseudo inverse. It approximates the solution of the equation system such that the sum of the squared errors is minimized.

4.2 Neural networks - architectural basics

As already described in the beginning of this chapter, deep neural networks are structures that consist of multiple layers of artificial neurons. In this case, they serve to perform complex regression tasks.
In the following sections, the inner workings of neural networks are explained, beginning with the very basic unit, the artificial neuron.

4.2.1 Artificial neurons

Artificial neurons consist of multiple inputs, which are weighted, summed up and used as input of the so called activation function. This results in the neuron's output.

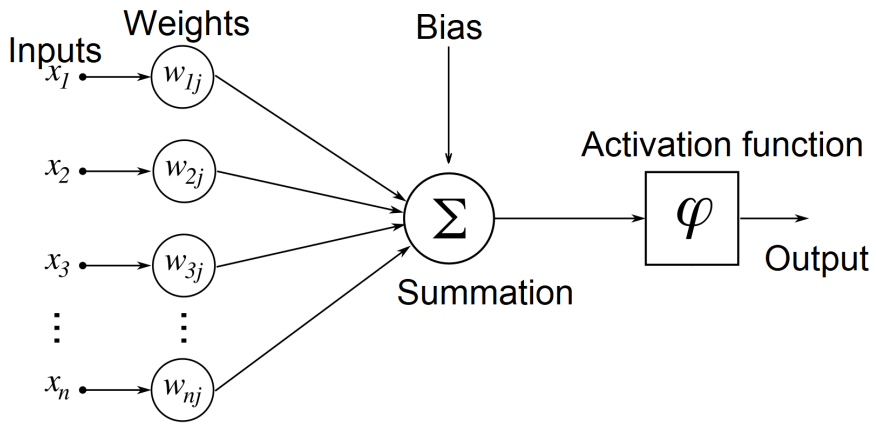


Figure 27: Illustration of an artificial neuron [16]

The activation function must be nonlinear. A linear function would result in a linear combination of input values, which does not transform the data in a useful way.
The bias is a special input of the neuron since it does not consist of a weighted input value. Rather, it is a constant that is added to each neuron's summation. Every neuron has it's own bias. Like the weights, they are parameters that need to be optimized.

4.2.2 Activation functions

Activation functions are nonlinear functions which determine the neuron's output. They have a strong influence on the network's performance. It is very advantageous if they have a defined derivative, because many learning algorithms need to derive them in order to work. Furthermore, they need to be monotonous. These are a few relevant ones:

Hard limit

This activation function sets the output to zero (or -1) if the weighted input sum is below the defined threshold. Otherwise, the output is set to one.
This function's advantage is its simplicity. However, the derivative is always zero, except at the threshold, where it is not defined.

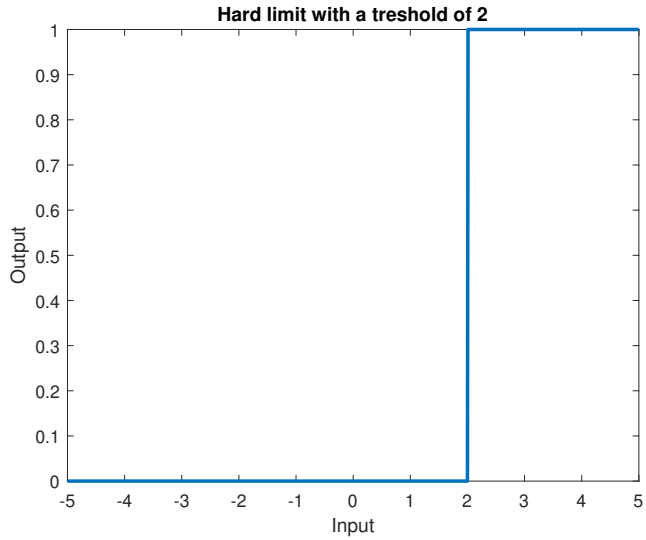


Figure 28: An example of a hard limit function

Hyperbolic tangent (tanh)

The hyperbolic tangent is also well suited to be used as an activation function. It is monotonous and has a useful (i.e. large enough) derivative for a wide range of input values. However, if the input is far away from zero, the gradient vanishes, which can become a problem in training. The hyperbolic tangent is defined as:

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{31}$$

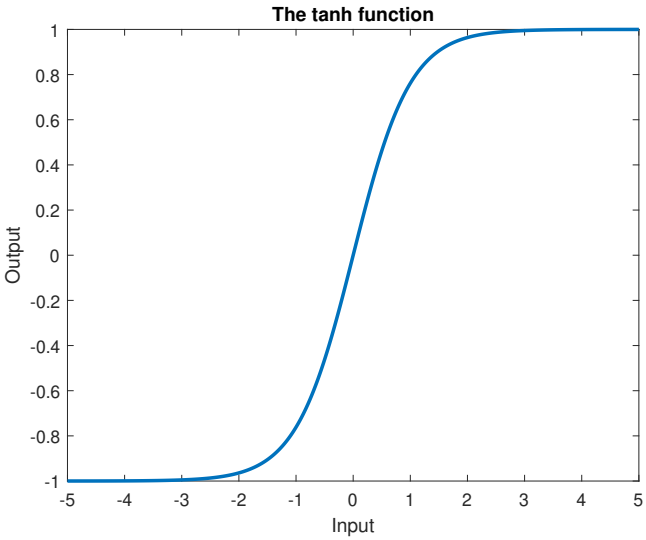


Figure 29: The hyperbolic tangent activation function

Sigmoid

The sigmoid looks very similar to the tanh function, but has a parameter a, which influences its behavior. It is very common and has good general properties, especially because the parameter a can be tuned to counter the vanishing gradient problem.

$$sig_a(x) = \frac{1}{1+e^{-ax}} \tag{32}$$

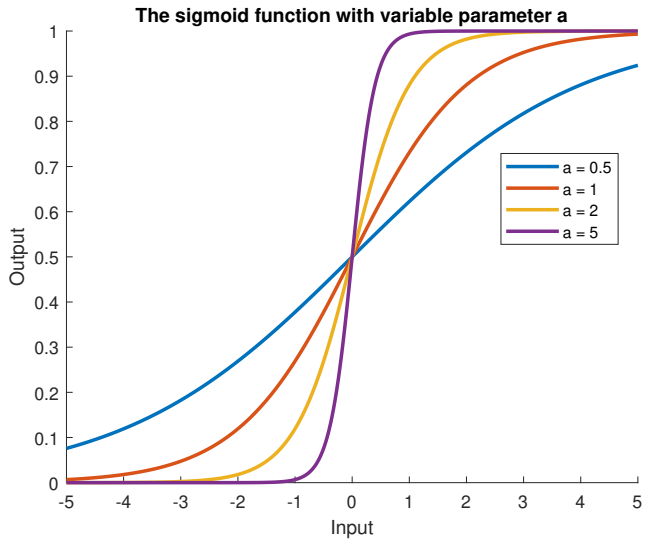


Figure 30: The sigmoid activation function with multiple values of the parameter a

Note that the sigmoid function maps the input to (0,1) as opposed to the tanh function, which maps to (-1,1).

Rectified linear unit (ReLU)

The rectified linear unit is a piecewise linear function, which puts out the input if it is positive and zero if it is negative:

$$ReLU(x) = x^+ = max(0,x) \tag{33}$$

Up until 2011, activation functions like the aforementioned sigmoid function were most commonly used in machine learning. In that year, it has been shown that the ReLU function actually performs better in computer vision tasks than the tanh or sigmoid functions [17]. Ever since, this somewhat counterintuitive function and it's extension, the leaky ReLU, have become more or less standard for many deep learning applications.

The obvious (see figure 31) disadvantage is that the function's derivative is zero for negative input values, which prevents learning if the occurring weighted input sums never reach a positive value. This is known as the dying ReLU problem [18], which can be tackled by inducing a parameter alpha, which determines the slope in the negative input region of the activation function. This approach is called the leaky ReLU function.

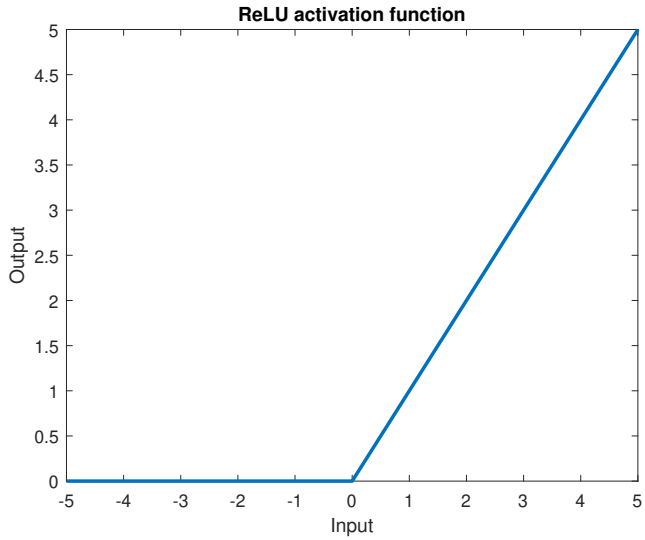


Figure 31: The ReLU activation function

Leaky ReLU

This extension enables the usage of the parameter α in order to also get a gradient in the negative input region. It is usually a better choice than the common ReLU function and commonly used in this thesis.

$$LReLU(x) = \begin{cases} x \cdot \alpha & x < 0 \\ x & x \geq 0 \end{cases} \tag{34}$$

As a side note, the derivative of (leaky) ReLU functions is not defined at $x = 0$. In practical applications, this is rather a technicality, but it can be a nuisance for theoretical derivations.

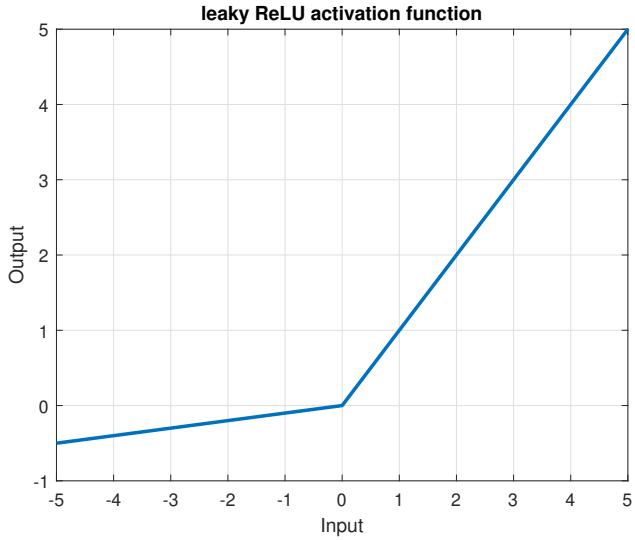


Figure 32: The leaky ReLU activation function (alpha = 0.1)

4.2.3 Approximation of arbitrary functions with neurons and overfitting

Before deep learning is tackled, another very important aspect of neural networks with one layer needs to be addressed: The "parallel" use of neurons enables an arbitrarily exact approximation of any given function in a defined interval. This is based on a similar idea to a series expansion, like Fourier or Taylor series. The target function is approximated by means of a superposition of differently scaled activation functions like sigmoids or ReLUs. This means that the neural network has one input neuron, one hidden layer with an arbitrary number of neurons and one output neuron. The main difference to series expansions is that the parameters are **not** calculated via an analytical expression like

$$a_k = \frac{1}{\pi} \int_{-\pi}^{\pi} f(t) \cdot cos(k \cdot t)\, dt \quad \forall k \geq 0 \tag{35}$$

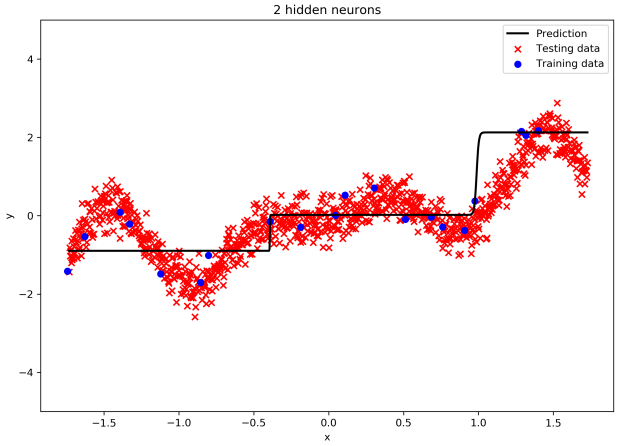
$$b_k = \frac{1}{\pi} \int_{-\pi}^{\pi} f(t) \cdot sin(k \cdot t)\, dt \quad \forall k \geq 1 \tag{36}$$

for the sine-cosine form of the Fourier series, but rather via learning algorithms, which iteratively improve the approximation's accuracy.

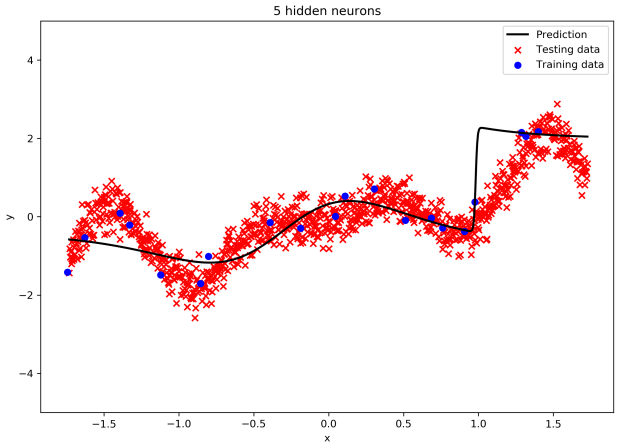
This works very well if the target function does not contain noise since the approximation will get better if the number of neurons is increased arbitrarily, like this would be the case for a series expansion with an ever increasing number of terms.

However, if the training data is noisy (as in practically all real world applications), this approach is limited by *overfitting*. This means that the approximation is fit "too well" to noisy data, which makes the behavior of the neural network less smooth and less able to generalize, like the following examples in figure 33 show [5], where a function is approximated by a combination of sigmoids. On the other hand, if the number of neurons is too low, *underfitting* occurs, see figure 33(a). Thus, there is a sweet spot that is to be found in order to optimize performance. This is also true for other machine learning applications, like deep neural networks.

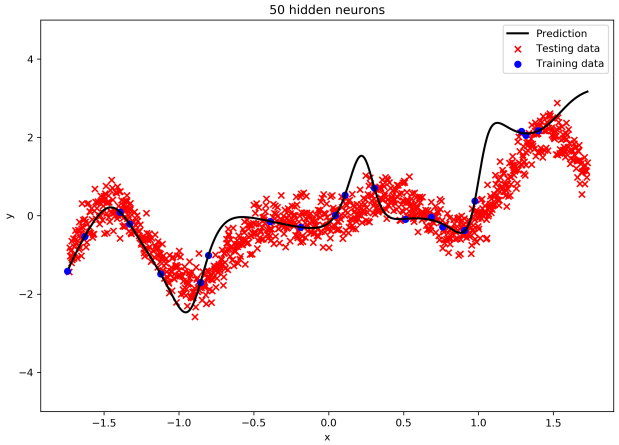
Note that in the given example, the number of training data points is relatively low compared to the number of testing data points (for more information see section on parameter training). This choice is by design since the scope of this example is to show the effects of overfitting. If the dataset was split such that the amount of training data is increased, the system would be less prone to overfitting. On the other hand, small testing data sets lead to a higher evaluation variance, but if the number of available data points is small, accepting this disadvantage should be considered.



(a) 2 hidden neurons lead to underfitting



(b) 5 hidden neurons work decently



(c) 50 hidden neurons lead to overfitting: The behavior becomes "wobbly" and represents the original function badly. This manifests in bad predictions, like the spike at x = 0.3.

Figure 33: Over- and underfitting when approximating a noisy function with sigmoid neurons [5]. Note that the blue points are training data points and the red crosses represent testing data, which are not taken into account for training and serve to evaluate the network's performance on "unseen" data.

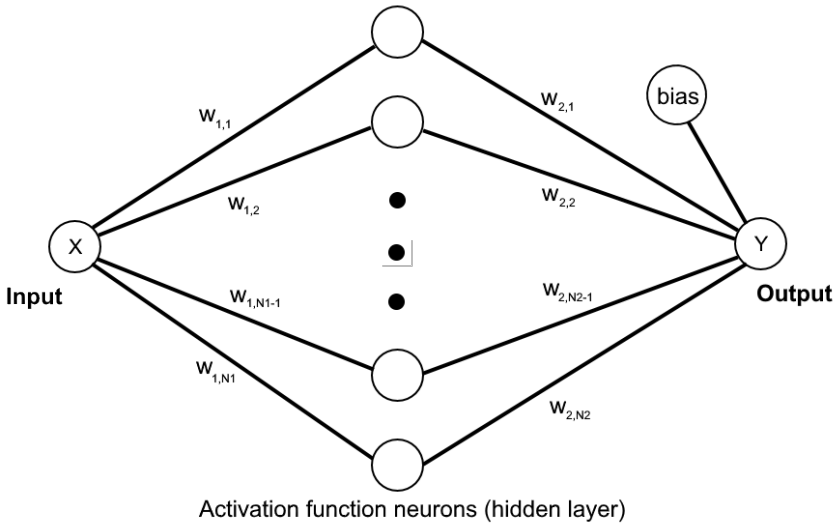


Figure 34: Simple neural network with one hidden layer and one input/output neuron. This architecture can be used to approximate functions, but is useless for any kind of artificial intelligence.

4.2.4 Deep neural networks

The next step on the way to an artificially intelligent system is to add more layers of neurons, see figure 35. This means that the outputs of one layer are connected to the inputs of the next one, which extends the network's ability to learn. For most regression tasks, this is an overkill, but for the one at hand, it proved to be a very solid approach, since the measurement values were not just composed of the path loss, but also by factors like the Rx and Tx antenna patterns, interfering signals from other nearby transmitter stations etc. Thus, some kind of "intelligence" is desirable.

One of the challenges of this approach is the huge amount of training parameters: Every line in figure 35 represents one parameter that is to be trained. It is obvious that more layers and neurons lead to an ever increasing computational effort.

Another undesirable property of deep neural networks is the fact that humans are in most cases hardly able to grasp what is going on in such a structure. This is a major difference to other signal processing approaches, which are usually well understood and understandable. Thus, deep learning can be seen as an art form, instead of an engineering discipline, because there is a lot of guessing and trial and error involved. This becomes even more apparent for more complex structures since they need a lot of time for training, which often makes parameter optimization strategies like grid searches unviable.

Another major challenge in deep learning is the fact that the iterative learning algorithms usually don't converge to global error minima and every training restart with different random starting parameters leads to different network performance. These aspects will be tackled in the following section.

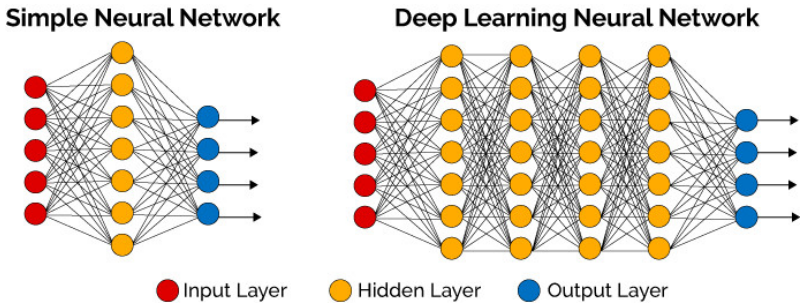


Figure 35: Comparison between simple neural networks and deep neural networks. The example on the left is similar to the one used in the previous chapter, except for the number of input and output neurons (5/4, as opposed to 1/1) [19]

4.3 Parameter training

The following section will explain how weights are updated, differentiating between the different types of gradient descent that exist. The whole part is based on [20]. As already mentioned, neural network parameters, i.e. the weights of the connections between the neurons and the biases $\mathbf{w}$, are trained by means of an iterative algorithm that approaches the minimum error step by step (usually, it converges to a local minimum). Figures 36 and 37 show graphical representations of gradients. After these, mathematical derivations follow.

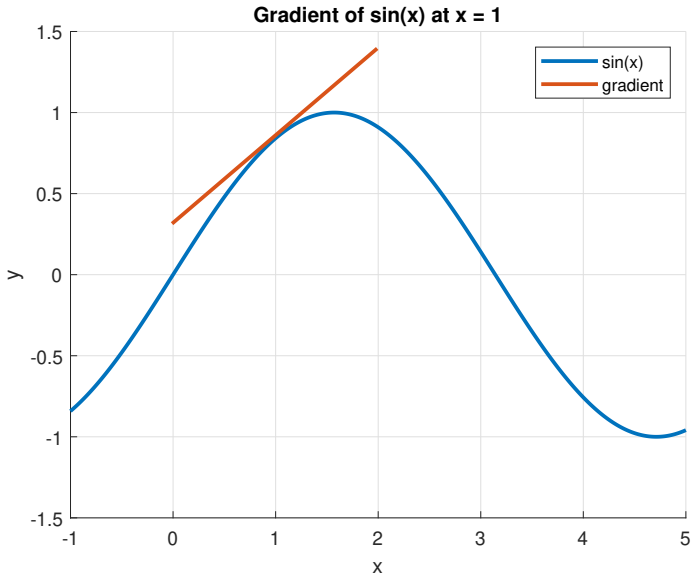
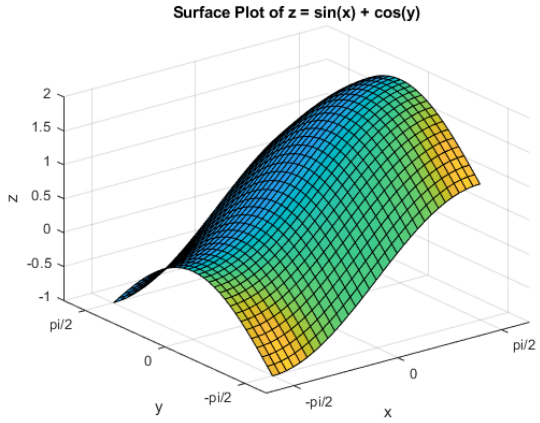
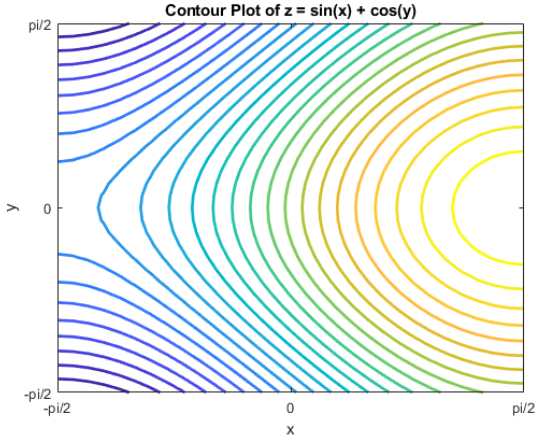


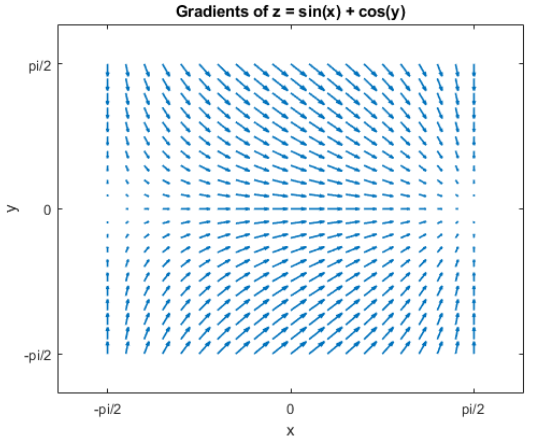
Figure 36: Gradient of a one dimensional sine function. In machine learning applications, the gradient operation is applied to multidimensional functions, sometimes reaching hundreds of thousands of dimensions. In this thesis, the dimensionality is in the range of hundreds.



(a) The three-dimensional surface plot of the simple function $z = sin(x) + cos(y)$



(b) The contour plot, which belongs to the above function



(c) The gradient field. The Arrows point towards the direction of the steepest ascend and their size represents the steepness.

Figure 37: Explanation of the gradient of a two-dimensional scalar field (the two input variables x and y are mapped to the scalar variable z). In machine learning applications, the cost function's scalar field's dimensionality easily exceeds hundreds, since every weight/bias adds one dimension.

The basis of gradient descent algorithms is the gradient of the cost function J with respect to the weights and biases $\nabla_{\mathbf{w}}(J)$. It contains the partial derivative of the cost function with respect to every weight/bias:

$$\nabla_{\mathbf{w}}(J) = \begin{pmatrix} \frac{\partial J}{\partial w_1} \\ \frac{\partial J}{\partial w_2} \\ \vdots \\ \frac{\partial J}{\partial w_N} \\ \frac{\partial J}{\partial b_1} \\ \frac{\partial J}{\partial b_2} \\ \vdots \\ \frac{\partial J}{\partial b_N} \end{pmatrix} \tag{37}$$

This vector is multiplied with the step size μ to get the weight updates. Since the gradient points towards the direction of the steepest ascent, it needs to be followed towards its negative direction for gradient descent.

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \mu \nabla_{\mathbf{w}}(J) \tag{38}$$

$$\begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_N \\ b_1 \\ b_2 \\ \vdots \\ b_N \end{pmatrix}_{k+1} = \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_N \\ b_1 \\ b_2 \\ \vdots \\ b_N \end{pmatrix}_{k} - \mu \begin{pmatrix} \frac{\partial J}{\partial w_1} \\ \frac{\partial J}{\partial w_2} \\ \vdots \\ \frac{\partial J}{\partial w_N} \\ \frac{\partial J}{\partial b_1} \\ \frac{\partial J}{\partial b_2} \\ \vdots \\ \frac{\partial J}{\partial b_N} \end{pmatrix} \tag{39}$$

In this approach, the gradient is calculated for every example in the training data set. All of these gradients are averaged, which results in the gradient that is used for for the weight update (equations 38 and 39). This is called **batch gradient descent**. A full iteration over the entire training data set is called an **epoch**. Thus, in batch gradient descent, one weight update is performed every epoch. This does not always have to be the case, as can be seen in the next subsection.

Types of gradient descent

There are three main types of gradient descent, which vary in the number of sample gradients that are summed up to determine the update "direction", i.e. the batch size. Figures 38(a) through 38(f) show the learning behavior for different batch sizes.

Batch gradient descent

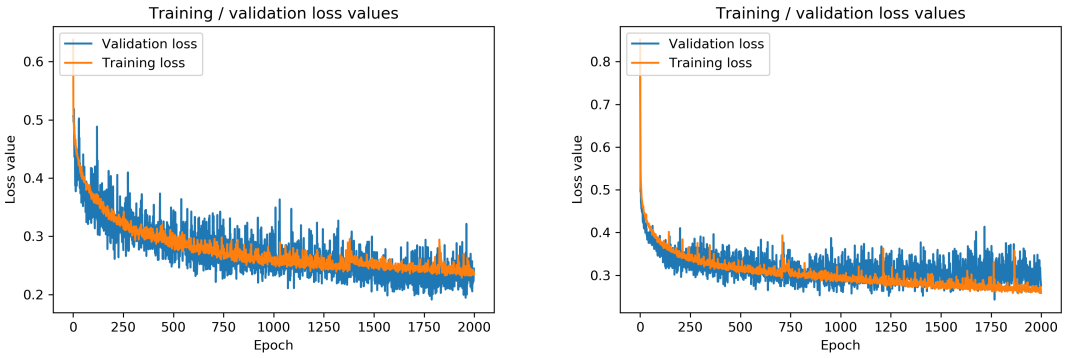
As already mentioned, this approach uses the average of the gradients from every sample in the test data set. Thus, one update step happens every epoch and the batch size is equal to the number of samples. This leads to a smooth and monotonic convergence towards a minimum. Of course, a lot of computation has to be done between every update step.

Stochastic gradient descent

This approach is the other extreme: After calculating the gradient for one sample, the weights are updated. This means that for every epoch, the number of updates is equal to the number of samples in the dataset. Of course, the convergence behavior will not be smooth. Actually, an update step may even lead to an increasing cost function, moving away from the minimum. In the long run however, the cost function is usually decreasing. This approach is useful for large datasets, which often occur in machine learning.

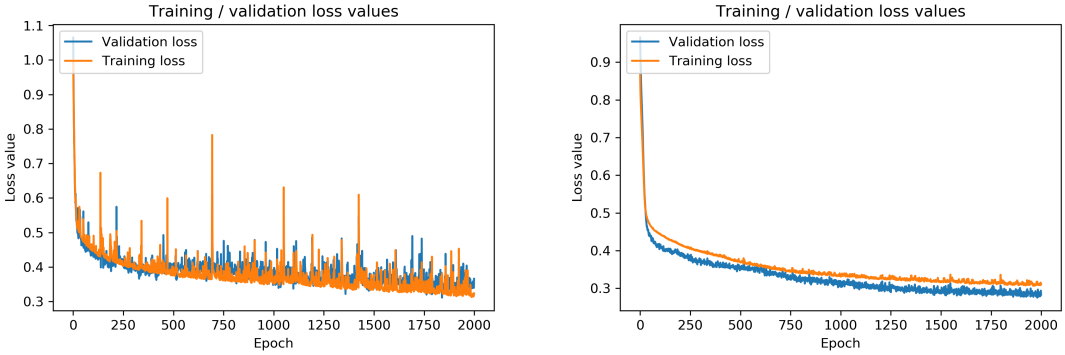
Mini Batch gradient descent

Of course, compromises between the two mentioned approaches can be found. Doing so is often the best way to go and also applied in this thesis. It is advisable to use a power of two as batch size because the learning algorithm can be optimized and sped up in this case. A significant advantage of smaller batch sizes is that their "twitchy" behavior may lead to better results since the optimizer might jump out of it's trajectory to a bad local minimum.



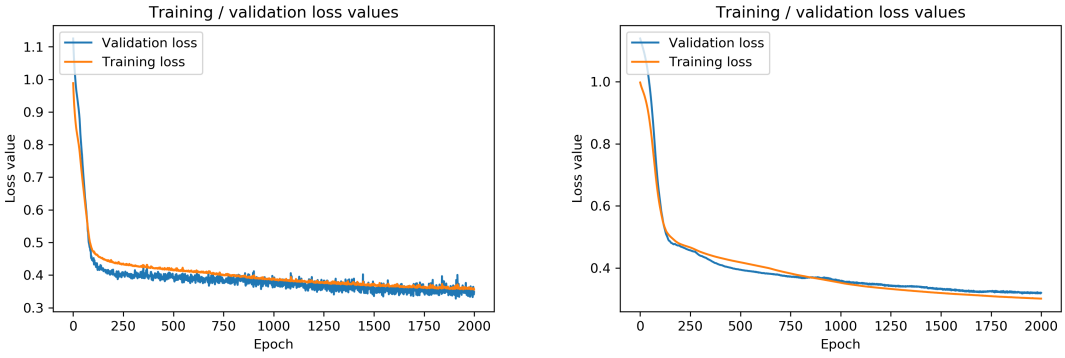
(a) Batch Size = 1

(b) Batch Size = 16



(c) Batch Size = 128

(d) Batch Size = 512



(e) Batch Size = 1024

(f) Batch Size = 8192

Figure 38: Comparison of learning procedures with different batch sizes. This figure serves as a qualitative explanation aid for the effects of different batch sizes. When it comes to quantitative performance evaluation, this is not a fair comparison since small batch sizes lead to more update steps within the 2000 Epochs (and take way longer to be computed).

4.4 Gradient determination or backpropagation

Determining the gradient of the cost function w.r.t. the entire weight/bias-vector in a deep neural network is not trivial since the effects of every one of these values are strongly interconnected. This section serves as a step-by-step summary of many ingredients of backpropagation, starting from a single neuron.

4.4.1 Gradient determination - single neuron

Up until now, the gradient of a neural network's cost function was used in many calculations, without going into details of its determination. In the case of multi-layer neural networks, there are actually a few challenges in this calculation.
To start off, the gradient calculation for a single neuron with respect to its weight is described.

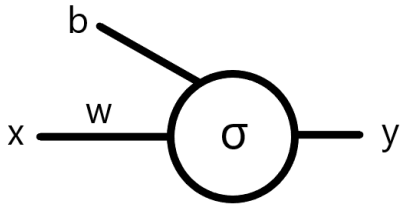


Figure 39: A single neuron and its necessary variables for gradient calculation

The first step is to set up the cost function J as the squared error, i.e. the difference between the neuron's actual output y and the target value t, with σ being the nonlinear neuron activation function.

$$J = (y(w) - t)^2 = (\sigma(wx + b) - t)^2 \tag{40}$$

$$J = \sigma^2(wx + b) - 2t\sigma(wx + b) + t^2 \tag{41}$$

The next step is to take the derivative of $J(w)$ with respect to w, also introducing the variable $z = wx + b$.

$$\frac{\partial J}{\partial w} = \frac{\partial}{\partial w}\left(\sigma^2(\underbrace{wx + b}_{z}) - 2t\sigma(\underbrace{wx + b}_{z}) + \underbrace{t^2}_{0}\right) \tag{42}$$

To further process this equation, the chain rule is applied:

$$\frac{\partial J}{\partial w} = \frac{\partial z}{\partial w} \cdot \frac{\partial y}{\partial z} \cdot \frac{\partial J}{\partial y} \tag{43}$$

Since $y = \sigma(z)$, the equation can be rewritten as:

$$\frac{\partial J}{\partial w} = x \cdot \frac{\partial \sigma(z)}{\partial z} \cdot 2(y - t) \tag{44}$$

4.4.2 Gradient determination - single layer parallel structure

In case of a single layer parallel structure, the output consists of the weighted sum of the neuron outputs.

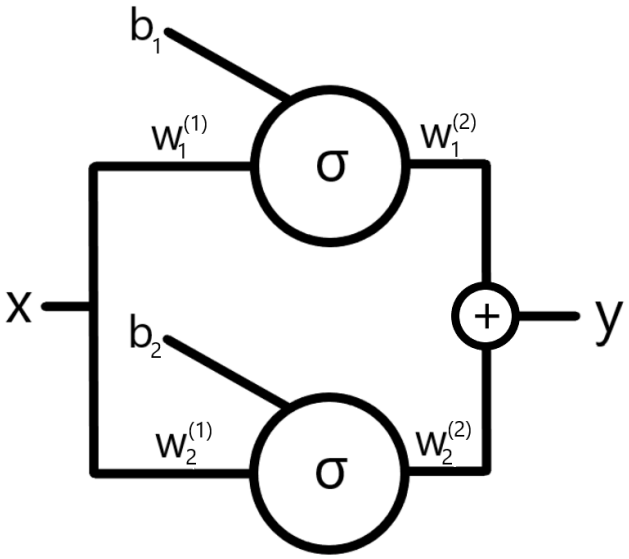


Figure 40: A parallel structure with N = 2 neurons

Thus, the previously scalar weight is now a vector. Furthermore, a second weight vector (for the summation) needs to be introduced.
This means that the column vector

$$\mathbf{w_1} = [w_1^{(1)} \;\; w_2^{(1)} \;\; \ldots \;\; w_N^{(1)}]^T \tag{45}$$

denotes the weights at the inputs of the nonlinear activation function, while the summation weights are described by

$$\mathbf{w_2} = [w_1^{(2)} \;\; w_2^{(2)} \;\; \ldots \;\; w_N^{(2)}]^T \tag{46}$$

Since there are N neurons in the net, N neuron input scalars z_n must exist. They are summarized as the row vector

$$\mathbf{z} = [z_1 \;\; z_2 \;\; \ldots \;\; z_N] \tag{47}$$

The numbers in the upper braces denote the layer that the weight appears in. **They are not exponents.**

For the derivation of the cost function's gradient, basically the same approach as before is used. If a vector form is desired, the cost function J can now be written as:

$$J = (y-t)^2 = (\sigma(\mathbf{z}) \cdot \mathbf{w_2} - t)^2 \tag{48}$$

With the replacement $z_n = w_{1,n} \cdot x + b_n$, the scalar summation form consists of

$$J = \left(\sum_{n=1}^{N} \sigma(w_n^{(1)} \cdot x + b_n) \cdot w_n^{(2)} - t \right)^2 \tag{49}$$

Same as before, the chain rule (see equation 43) is applied in order to get an analytic expression for the gradient of the cost function w.r.t the neuron input weights $\mathbf{w_1}$. This time however, some adaptations have to be taken into account:

$$z_n = w_n^{(1)} \cdot x + b_n \Longrightarrow \frac{\partial z_n}{\partial w_n^{(1)}} = x \tag{50}$$

$$\frac{\partial y}{\partial \mathbf{z}} = \frac{\partial}{\partial \mathbf{z}} \sum_{n=1}^{N} w_n^{(2)} \cdot \sigma(z_n) = \sum_{n=1}^{N} w_n^{(2)} \frac{\partial \sigma(z_n)}{\partial z_n} \tag{51}$$

With the equation for $\frac{\partial J}{\partial y}$ being the same as before, the resulting derivative becomes

$$\frac{\partial J}{\partial \mathbf{w^{(1)}}} = \sum_{n=1}^{N} w_n^{(2)} \cdot \frac{\partial \sigma(z_n)}{\partial z_n} \cdot x \cdot 2(y - t) \tag{52}$$

The derivative of the cost function with respect to the addition layer weights is equal to the respective inputs, since they determine the sensitivity of the gradient towards weight changes:

$$\frac{\partial J}{\partial \mathbf{w^{(2)}}} = \sigma(\mathbf{z}) \tag{53}$$

4.4.3 Gradient determination - multi layer parallel structure

The next step on the way to understanding backpropagation calculus is to look at the derivatives of the cost function for a multilayer structure like a deep neural network. Intuitively, it makes sense that weights that appear in a later layer of the network (say $w_3^{(3)}$ will influence the gradient's sensitivity with respect to one of the previous weights (say $w_1^{(1)}$). This effect can also be observed in the previous structure, where $\frac{\partial J}{\partial \mathbf{w}^{(1)}}$ is influenced by the weights of the addition layer, see equation 52.
Actually, this structure is not significantly more complex than the previous one. How the gradient for a single layer is computed, is already known at this point. The main difference is that the expressions for $z_n^{(2)}$, i.e. the summed inputs of the second layer neurons, have to take into account more terms, namely the outputs of the previous layer y_m and the weights $w_{m,n}$ that connect them to the neuron:

$$z_n^{(2)} = \sum_{m=1}^{N^{(1)}} y_m \cdot w_{m,n} + b_n^{(n)} \tag{54}$$

In equation 54, $N^{(1)}$ denotes the number of neurons in the first layer.
If the gradient w.r.t $\mathbf{w^{(2)}}$ is sought, the following expression can be used:

$$\frac{\partial J}{\partial \mathbf{w^{(2)}}} = \frac{\partial J}{\partial \mathbf{y^{(2)}}} \frac{\partial \mathbf{y^{(2)}}}{\partial \mathbf{z^{(2)}}} \frac{\partial \mathbf{z^{(2)}}}{\partial \mathbf{w^{(2)}}} \tag{55}$$

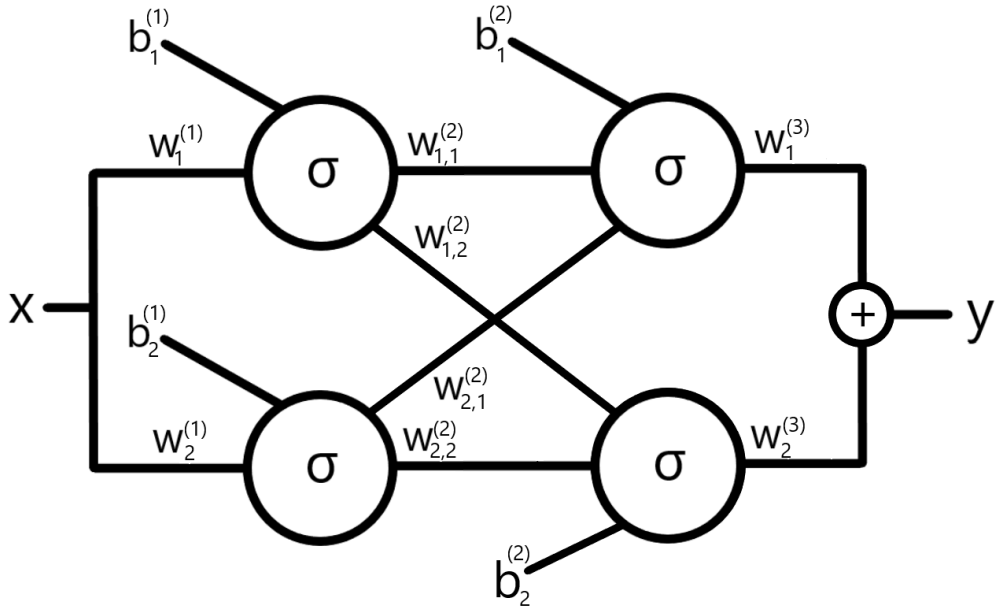


Figure 41: A parallel structure with two layers. The number of neurons in the first layer is equal to the number of neurons in the second layer: $N_1 = N_2 = 2$

The gradient w.r.t $\mathbf{w^{(1)}}$ can be calculated by a further extension:

$$\frac{\partial J}{\partial \mathbf{w}^{(\mathbf{1})}} = \frac{\partial J}{\partial \mathbf{y}^{(\mathbf{2})}} \frac{\partial \mathbf{y}^{(\mathbf{2})}}{\partial \mathbf{z}^{(\mathbf{2})}} \frac{\partial \mathbf{z}^{(\mathbf{2})}}{\partial \mathbf{y}^{(\mathbf{1})}} \frac{\partial \mathbf{y}^{(\mathbf{1})}}{\partial \mathbf{w}^{(\mathbf{1})}} \tag{56}$$

Looking at equations 55 and 56, it becomes apparent why the algorithm is called backpropagation.
Since the calculation of the gradients starts getting too laborious and it is not the main scope of this thesis, these explanations are halted at this point. If the vector calculations are generalized such that an arbitrary number of layers and neurons in every layer can be chosen, the matrix form of the backpropagation algorithm is reached. More on this topic can be found at [21].

5 MATLAB scripts - linear regression

The linear regression part of the thesis was conducted in MATLAB since it is relatively easy to use and has all the functionality needed for the tasks at hand. The following chapter discusses the most relevant code snippets. Full codes can be found in the appendix; the basic terminology and approach is detailed in chapter 3.

The main scripts for linear regression are called `Analysis.m` and `Automated_Analysis.m`, with the first one only conducting one linear regression analysis for one corridor and the second one iterating through a defined range of elevation angles. Thus, the first script will be looked at primarily as it suffices for an explanation of the concepts used in this part of the thesis.

5.1 Loading of flight data

The first step is loading the flight data. The relevant parameter, i.e. the file name, is set around line 40 because the beginning of the script was reserved for setting the corridor that the linear regression should be performed for. This task is conducted by an automatically generated part of the script that loads the entries of a csv-file.

```
39 %% Initialize variables.
40 filename = '.\DirnbachConverted\CSV\865116045125976_8_10_2020_12.5.6.azm.csv';
41 delimiter = ',';
42 startRow = 2;
```

The import script automatically generated by MATLAB ends at line 96. This code puts a data frame, simply called `Dataframe` into the workspace, see figure 42. The elements of this data frame are then easily accessed with lines like `RSRP = Dataframe.rsrp1;`. Positional information is stored by means of geocoordinates, which need to be converted to Cartesian and polar coordinates, centered around the transmitter station. The formulas used for this task are linear approximations for this transform.

```
97  %% Start of actual analysis script
98  % by Giancarlo Benincasa
99  % 11/2020
100 % Data from: 865116045125976_8_10_2020_12.5.6.azm.csv
101
102 % get RSRP and RSRQ
103 RSRP = Dataframe.rsrp1;
104 RSRQ = Dataframe.rsrq1;
105
106 % get postitional information (geocoordinates/height above sea level)
107 altitude = Dataframe.altitude;
108 latitude = Dataframe.latitude;
109 longitude = Dataframe.longitude;
110
111 %Transmitter Station coordinates
```

```
LongitudeTx = 15.892169;
LatitudeTx = 46.830863;

% convert them to x,y,z
x = (longitude - LongitudeTx)*76000;
y = (latitude - LatitudeTx)*111000;
z = altitude-min(altitude);

% Coordinate transforms
[azimuth,elevation,r] = cart2sph(x,y,z);
azimuth = azimuth*180/pi;
elevation = elevation*180/pi;

% Polar coordinate matrix
X = [azimuth,elevation,z]';
```

The script plots the entire flight and performs a histogram analysis of the angles, see figure 43.

Editor - Analysis.m | Variables - Dataframe

Dataframe

2823x22 table

	1 time	2 longitude	3 latitude	4 altitude	5 height	6 rsrp1	7 sinr1	8 pci1	9 cgi	10 rsrq1	11 pci2	12 rsrp2	1 sir
2	"2020-10-08...	15.8914	46.8372	294.7754	-2.9184	-76.1208	19.8000	369	2.5509e+14	-5.6042	NaN	NaN	
3	"2020-10-08...	15.8914	46.8372	294.7754	-2.9184	-76.2383	18.6000	369	2.5509e+14	-5.7070	NaN	NaN	
4	"2020-10-08...	15.8913	46.8372	301.6389	3.9451	-76.3148	15.8185	369	2.5509e+14	-6.4398	NaN	NaN	
5	"2020-10-08...	15.8913	46.8372	301.6389	3.9451	-74.5060	12.6774	369	2.5509e+14	-7.2621	NaN	NaN	
6	"2020-10-08...	15.8913	46.8372	301.6389	3.9451	-80.4507	17.5474	369	2.5509e+14	-8.7138	NaN	NaN	
7	"2020-10-08...	15.8913	46.8372	301.6389	3.9451	-78.3125	14.7619	369	2.5509e+14	-7.4821	NaN	NaN	
8	"2020-10-08...	15.8914	46.8372	300.7203	3.0265	-76.5881	13.7182	369	2.5509e+14	-7.3551	NaN	NaN	
9	"2020-10-08...	15.8914	46.8372	300.7203	3.0265	-75.2688	9.7300	369	2.5509e+14	-7.2344	NaN	NaN	
10	"2020-10-08...	15.8914	46.8372	300.7203	3.0265	-73.6625	17.5640	369	2.5509e+14	-6.6300	NaN	NaN	
11	"2020-10-08...	15.8914	46.8372	301.2593	3.5654	-79.3494	21.4000	369	2.5509e+14	-6.9091	NaN	NaN	
12	"2020-10-08...	15.8914	46.8372	301.2593	3.5654	-74.4464	17.8952	369	2.5509e+14	-2.3571	NaN	NaN	
13	"2020-10-08...	15.8914	46.8372	301.2593	3.5654	-82.5938	23.1500	369	2.5509e+14	-7.4344	NaN	NaN	
14	"2020-10-08...	15.8914	46.8372	301.2593	3.5654	-85.1838	18.7941	369	2.5509e+14	-7.6801	NaN	NaN	
15	"2020-10-08...	15.8914	46.8372	301.2593	3.5654	-85.2143	19.0476	369	2.5509e+14	-7.9673	NaN	NaN	
16	"2020-10-08...	15.8914	46.8372	301.4890	3.7952	-82.0370	13.9593	369	2.5509e+14	-6.7269	NaN	NaN	
17	"2020-10-08...	15.8914	46.8372	301.4890	3.7952	-81.9258	12.4125	369	2.5509e+14	-7.3242	NaN	NaN	
18	"2020-10-08...	15.8914	46.8372	301.4890	3.7952	-81.4844	7.8917	369	2.5509e+14	-6.9115	NaN	NaN	
19	"2020-10-08...	15.8914	46.8372	301.4890	3.7952	-80.5085	12.5773	369	2.5509e+14	-6.3125	NaN	NaN	
20	"2020-10-08...	15.8914	46.8372	301.4199	3.7261	-81.1528	12.7222	369	2.5509e+14	-6.3403	NaN	NaN	
21	"2020-10-08...	15.8914	46.8372	301.4199	3.7261	-81.5156	15.4000	369	2.5509e+14	-6.5000	NaN	NaN	
22	"2020-10-08...	15.8914	46.8372	301.4199	3.7261	-81.7500	10.1571	369	2.5509e+14	-6.6786	NaN	NaN	
23	"2020-10-08...	15.8914	46.8372	301.4199	3.7261	-82.5804	15.6500	369	2.5509e+14	-6.8259	NaN	NaN	

Figure 42: The automatically generated data frame

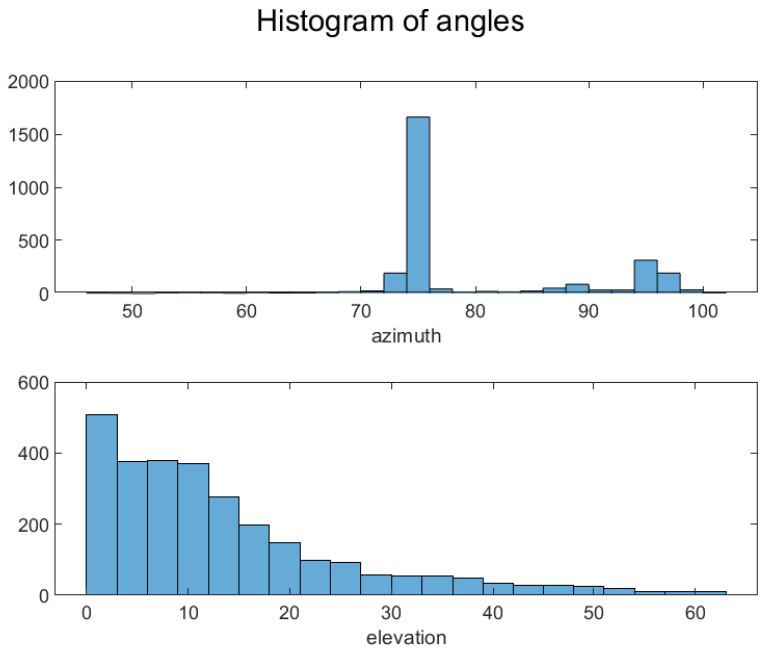


Figure 43: The histogram angle analysis. This serves to determine in which angle intervals the wall can be found.

5.2 Setting the corridor

As already mentioned, the script `Analysis.m` only analyzes one corridor. To choose possible angles, one has to have a look at the histogram, see figure 43. Obviously, the wall was flown in an azimuth range from about 73 to 76 degrees. The number of data points strongly diminishes for higher elevation angles, such that a corridor between ten and eleven degrees is chosen for this explanation. Since the angular values get tweaked a lot, they are defined right at the beginning of the script:

```
clc
clear all
close all

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Data from: 865116045125976_8_10_2020_12.5.6.azm.csv
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                ANALYSIS PARAMETERS
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Select "Corridor" between elevation_min and elevation_max
elevation_min = 10;
elevation_max = 11;
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Select "Corridor" between azimuth_min and azimuth_max
% Better not change these, once properly selected
azimuth_min = 73;
azimuth_max = 76;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

These settings are then used in the portion of the code that throws away all data points that are outside of the defined region:

```
% throw away RSRP/RSRQ data that stems from > azimuth_max
t_sel = t;
azimuth_sel = azimuth;
elevation_sel = elevation;
r_sel = r;
x_sel = x;
y_sel = y;
z_sel = z;
t_sel(azimuth > azimuth_max) = [];
azimuth_sel(azimuth > azimuth_max) = [];
elevation_sel(azimuth > azimuth_max) = [];
r_sel(azimuth > azimuth_max) = [];
x_sel(azimuth > azimuth_max) = [];
y_sel(azimuth > azimuth_max) = [];
z_sel(azimuth > azimuth_max) = [];

% throw away < azimuth_min
t_sel(azimuth_sel < azimuth_min) = [];
elevation_sel(azimuth_sel < azimuth_min) = [];
r_sel(azimuth_sel < azimuth_min) = [];
x_sel(azimuth_sel < azimuth_min) = [];
y_sel(azimuth_sel < azimuth_min) = [];
z_sel(azimuth_sel < azimuth_min) = [];
azimuth_sel(azimuth_sel < azimuth_min) = [];


% throw away > elevation_max
t_sel(elevation_sel > elevation_max) = [];
azimuth_sel(elevation_sel > elevation_max) = [];
r_sel(elevation_sel > elevation_max) = [];
x_sel(elevation_sel > elevation_max) = [];
y_sel(elevation_sel > elevation_max) = [];
z_sel(elevation_sel > elevation_max) = [];
elevation_sel(elevation_sel > elevation_max) = [];

% throw away < elevation_min
t_sel(elevation_sel < elevation_min) = [];
azimuth_sel(elevation_sel < elevation_min) = [];
r_sel(elevation_sel < elevation_min) = [];
x_sel(elevation_sel < elevation_min) = [];
y_sel(elevation_sel < elevation_min) = [];
```

```
212  z_sel(elevation_sel < elevation_min) = [];
213  elevation_sel(elevation_sel < elevation_min) = [];
```

At this point, the corridor is selected and preparations for linear regression are almost completed. The variable `r_sel` contains all selected radii and `t_sel` all selected target (i.e. RSRP or RSRQ) values. However, since linear behavior is expected on a log-log plot, the radii need to be transformed accordingly. Letting the x-axis start at zero would not make any sense because zero on a linear scale corresponds to $-\infty$ logarithmically. Thus, d_0 is defined as 100 meters:

```
236  % TEST OF LOGLOG MODEL
237  r_sel_log = log10(r_sel/100);            % d0 = 100m
```

The next step is the calculation of the linear regression by means of the Moore-Penrose pseudo inverse (least squares cost function, see chapter 4.1). This is easily done in MATLAB by simply using a backslash, see line 241:

```
239  % calculate linear regression
240  kd_mat = [ones(length(r_sel_log),1) r_sel_log];
241  kd = kd_mat\t_sel;
242  t_linreg = kd_mat*kd;
```

The matrix `kd` contains the calculated offset and slope of the linear regression function. `t_linreg` contains values predicted by the regression for the selected radii.

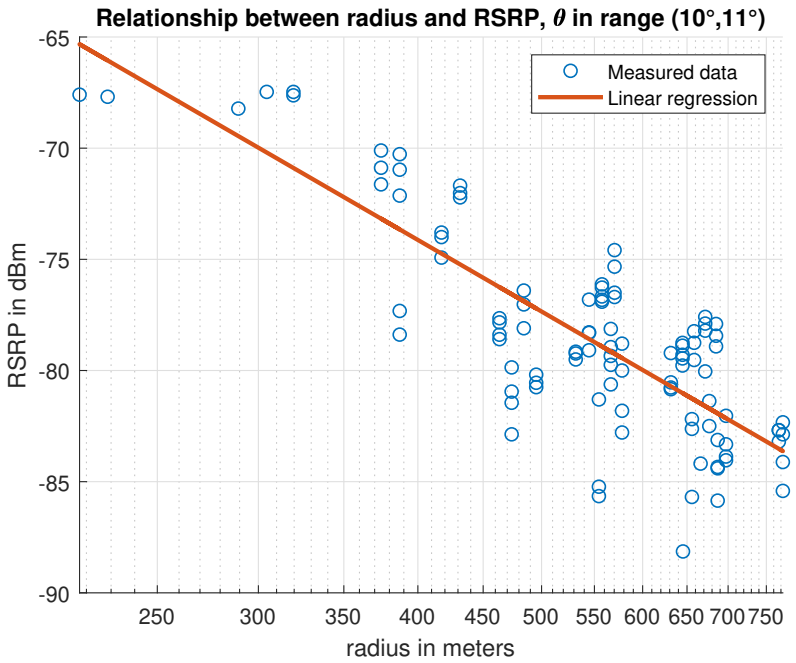


Figure 44: Linear regression (orange line) and measurements (blue circles) for the example at hand

5.3 Automated analysis

Now that the basic principles of the script's inner workings are explained, the last step is to have a look at the automated analysis in the MATLAB file `Automated_Analysis.m`. The main difference is that the corridors that are getting iterated through now need to be defined as vectors in the beginning of the script. Furthermore, the target definition can also be found in an earlier line (28) of the script:

```
1  clc
2  clear all
3  close all
4
5  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
6  % Data from: 865116045125976_8_10_2020_12.5.6.azm.csv
7  % can be changed though
8  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
9
10 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
11 %       USE THIS SCRIPT TO ANALYZE RSRP FOR
12 %       MULTIPLE ELEVATION ANGLES AT ONCE
13 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
14
15 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
16 %               ANALYSIS PARAMETERS
17 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
18 % Select "Corridors" between elevation_min and elevation_max
19 elevation_min = 4:20;
20 elevation_max = 5:21;
21 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
22 % Select "Corridor" between azimuth_min and azimuth_max
23 % Better not change these, once properly selected
24 azimuth_min = 73;
25 azimuth_max = 76;
26 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
27 % set target (RSRP or RSRQ)
28 target = "RSRP"
```

Then, the script also filters out the necessary azimuth range. When it comes to filtering elevation angles, there is a major difference, as this process is now conducted within a loop, calculating the regression in every iteration.

```
234 for ii = 1:1:length(elevation_max)
235
236     t_sel = t_cleaned;
237     azimuth_sel = azimuth_cleaned;
238     elevation_sel = elevation_cleaned;
239     r_sel = r_cleaned;
240     x_sel = x_cleaned;
241     y_sel = y_cleaned;
242     z_sel = z_cleaned;
```

```
243
244     % throw away > elevation_max
245     t_sel(elevation_sel > elevation_max(ii)) = [];
246     azimuth_sel(elevation_sel > elevation_max(ii)) = [];
247     r_sel(elevation_sel > elevation_max(ii)) = [];
248     x_sel(elevation_sel > elevation_max(ii)) = [];
249     y_sel(elevation_sel > elevation_max(ii)) = [];
250     z_sel(elevation_sel > elevation_max(ii)) = [];
251     elevation_sel(elevation_sel > elevation_max(ii)) = [];
252
253     % throw away < elevation_min
254     t_sel(elevation_sel < elevation_min(ii)) = [];
255     azimuth_sel(elevation_sel < elevation_min(ii)) = [];
256     r_sel(elevation_sel < elevation_min(ii)) = [];
257     x_sel(elevation_sel < elevation_min(ii)) = [];
258     y_sel(elevation_sel < elevation_min(ii)) = [];
259     z_sel(elevation_sel < elevation_min(ii)) = [];
260     elevation_sel(elevation_sel < elevation_min(ii)) = [];
261
262     % LOGLOG MODEL
263     r_sel = log10(r_sel/100);
264
265     % calculate linear regression
266     kd_mat = [ones(length(r_sel),1) r_sel];
267     kd = kd_mat\t_sel;
268     RSRP_linreg = kd_mat*kd;
269
270     % save data
271     kd_saved = [kd_saved kd];
272     angles_saved = [angles_saved (elevation_min(ii)+elevation_max(ii))/2];
273     nr_datapoints_saved = [nr_datapoints_saved length(r_sel)];
274
275     % plot regression and elevation angle
276     plot(100*10.^(r_sel),(kd(1)+r_sel*kd(2)),"LineWidth",1.2)
277     txt = num2str(elevation_max(ii));
278     text(100*10.^(r_sel(1)),(kd(1)+r_sel(1)*kd(2)),txt)
279
280 end
281
282
283 % calculate mean k and d values, weighted by number of datapoints
284 kmean = sum(kd_saved(2,:).*nr_datapoints_saved(:).')/sum(nr_datapoints_saved);
285 dmean = sum(kd_saved(1,:).*nr_datapoints_saved(:).')/sum(nr_datapoints_saved);
286 %plot weighted mean regression line
287 plot((min(r):max(r)),(dmean+log10((min(r):max(r))/100)*kmean),"LineWidth",4,"Color","red")
288 txt = "weighted mean";
289 text(123,-60,txt)
290 set(gca,'xscale','log')
```

The script also plots the calculated linear regression in every iteration. In the end, the mean, weighted by the number of data points, is calculated and plotted as a thick red line.

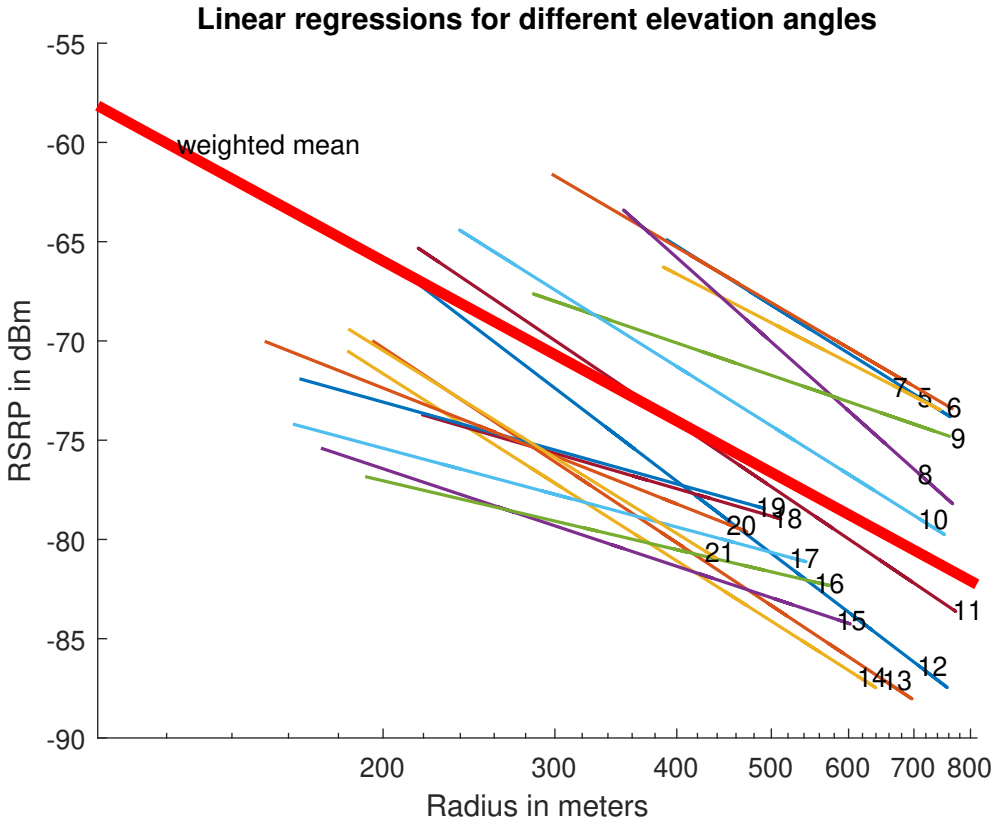


Figure 45: Calculated linear regressions and their weighted mean

The results, i.e. the linear equation coefficients and the weighted means, are also summarized in a plot vs. the elevation angles that they belong to (figure 46).

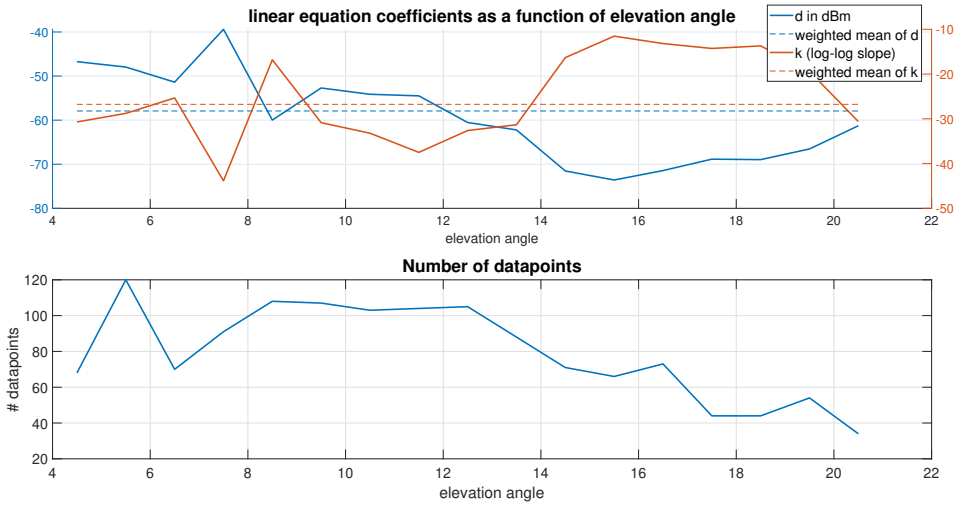


Figure 46: Calculated linear regression parameters vs. elevation angle and their weighted means (dashed lines)

6 Python scripts - machine learning

In this chapter, the most important parts of the machine learning scripts are presented. This part of the thesis was written in Python since it enables the use of Google's Tensorflow library.
In this summary, larger portions of the code are left out than in chapter 5. This is due to the fact that most of the work in Python actually consisted of data preparation and pre-processing. The entire Jupyter Notebooks used in this thesis can be found in the appendix.
The data preparation regarding data transformation from geocoordinates to Cartesian/polar ones is done by basically the same steps as in the MATLAB scripts. The following table shows the used libraries and their respective purposes:

Library	Purpose
pandas	data analysis, CSV processing
NumPy	array and matrix computations
math	mathematical functions
TensorFlow, Keras	machine learning, neural networks
Seaborn	autmated statistical analysis
random	random number generation
Matplotlib	plotting of results

One of the most important factors influencing machine learning performance is the separation of measurements into training and test data. This is done using a pseudo random separation with a defined seed to enable reproducible results. Thus, there are two main python scripts (or rather, ipython notebooks) used in this thesis: `MA_v2.ipynb` and `MA_v3_optimizer.ipynb`. The latter iterates through possible separation seeds and lets the net learn for 2000 epochs. Results are then plotted and saved in a folder.
This approach enables the selection of a seed that leads to good results (a better local minimum is found) and using this seed for longer learning in the `MA_v2.ipynb` script.

6.1 Preprocessing of data

After having imported the necessary libraries, i.e. `pandas`, `numpy` and `math`, and with the data import/-transformation out of the way, the next step is to determine which transmitter station is connected to the mobile at any given measurement point. This is necessary because - other than in the previous chapter - the measurements contain connections to different transmitter stations and are not limited to just one.
This part is contained in both scripts. The line numbers correspond to those in `MA_v2.ipynb`.

```
79  # create dict of raw cell IDs
80  cellIDs = meas_df.RAWCELLID
81  IDnumber = 0 #number that is assigned to each unique cell ID, starting with zero
82  IDdict = {} #dictionary containing cell IDs and their respective numbers
83  for i in range(len(cellIDs)):
84      if not cellIDs[i] in IDdict:
```

```
85            IDdict[cellIDs[i]] = IDnumber
86            IDnumber = IDnumber + 1
87    IDnumberMax = IDnumber
88    print("ID dictionary:")
89    print(IDdict)
90
91    # create array of ID numbers
92    #IDnumberArray = np.zeros(len(cellIDs))    # np array
93    IDnumberArray = []                         # python array
94    for i in range(len(cellIDs)):
95        #IDnumberArray[i] = int(IDdict[cellIDs[i]])      # np array
96        IDnumberArray.append(int(IDdict[cellIDs[i]]))    # python array
97
98    # check for number of station appearances
99    stationCount = 0
100   for j in range(IDnumberMax):
101       for i in range(len(cellIDs)):
102           if IDnumberArray[i] == j:
103               stationCount = stationCount + 1
104       print("IDnumber "+str(j)+" appears "+str(stationCount)+" times.")
105       stationCount = 0
```

Figure 47 shows a possible output of the data preprocessing script. Since RSRP/RSRQ prediction makes the most sense if all measurements are conducted such that the signal is received from only one transmitter station, the script contains the option to perform this filtering:

```
130   # OPTIONAL: DUMP ALL DF ENTRIES EXCEPT FOR THOSE OF A CERTAIN TX STATION
131   id_sel = 0 # set to -1 if no ID should be selected
132   if id_sel != -1:
133       df = df[df['ID number'] == id_sel]
134       df = df.drop(columns=['ID number'])  # ID number column no longer needed
135       print('\n ID number '+str(id_sel)+' selected. All other entries dumped. \n')
```

After these steps, the data is preprocessed and the actual neural network training can be conducted.

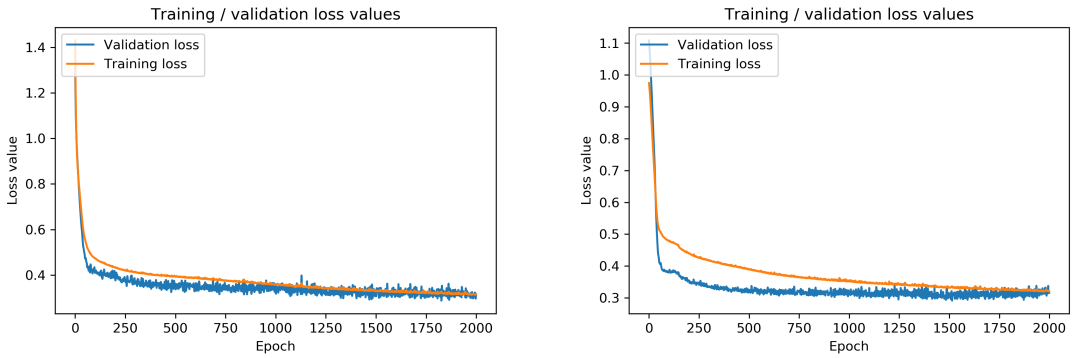
```
ID dictionary:
{20279051: 0, 20279071: 1, 18018336: 2, 19700512: 3, 18018316: 4, 20279061: 5, 19700492: 6, 18383884: 7, 18383904: 8, 20279
072: 9, 18384908: 10, 25127180: 11, 19400459: 12, 19319564: 13, 18383894: 14, 19417121: 15, 20279062: 16, 20279052: 17}
IDnumber 0 appears 2355 times.
IDnumber 1 appears 342 times.
IDnumber 2 appears 104 times.
IDnumber 3 appears 12 times.
IDnumber 4 appears 30 times.
IDnumber 5 appears 26 times.
IDnumber 6 appears 34 times.
IDnumber 7 appears 286 times.
IDnumber 8 appears 109 times.
IDnumber 9 appears 27 times.
IDnumber 10 appears 167 times.
IDnumber 11 appears 26 times.
IDnumber 12 appears 40 times.
IDnumber 13 appears 4 times.
IDnumber 14 appears 10 times.
IDnumber 15 appears 6 times.
IDnumber 16 appears 4 times.
IDnumber 17 appears 15 times.

 ID number 0 selected. All other entries dumped.

            count        mean         std         min         25%         50%  \
x          2354.0   34.035011  127.898502 -313.657854  -37.359854   59.312146
y          2354.0  -48.302426  153.347588 -328.426251 -184.320501  -38.272251
z          2354.0  106.471113   43.175270    0.000000   81.000000  109.000000
azimuth    2354.0    2.029133   86.587235 -179.860919  -51.891158  -40.682334
elevation  2354.0   34.431965   18.967678    0.000000   19.502566   29.626194
radius     2354.0  217.380682   96.421345    3.580483  136.642691  204.301087
RSRP       2338.0  -86.479469    5.321810  -98.000000  -90.000000  -87.000000
RSRQ       2338.0  -11.600941    2.955437  -20.000000  -14.000000  -12.000000

                 75%         max
x          124.159146  255.658146
y           61.627749  333.577749
z          137.000000  189.000000
azimuth     84.477276  179.833687
elevation   45.294811   85.406940
radius     288.281221  450.488563
RSRP       -83.000000  -68.000000
RSRQ        -9.000000   -5.000000
Reading and preprocessing of data done
```

Figure 47: One possible output of the data preprocessing script. Most measurements stem from connections to the transmitter station with ID 0; all other measurements have been dumped.

6.2 Finding a good data splitting seed

As already discussed, the seed used for random data splitting strongly influences the results. Thus, training (with 2000 epochs as opposed to 10000 in actual training) is repeated in a loop with a starting seed of 0, which is incremented in every iteration. Then, the training results (i.e. training loss and validation loss behavior) are saved, which enables a manual selection of a decent seed. Two of these results are depicted in figures 48(a) and 48(b).

This section also serves as an explanation of the machine learning script. Basically, the only difference between `MA_v2.ipynb` and `MA_v3_optimizer.ipynb` is the fact that the latter contains a loop that repeats the training process. Thus, an explanation of this script suffices for an understanding of the code.



(a) Random seed 0

(b) Random seed 14

Figure 48: Different seeds lead to different loss functions as different local minima of the cost function are approached (Prediction of RSRP, spherical coordinates).

The first major piece of code imports all necessary libraries, selects coordinates (spherical/Cartesian) and target (RSRP/RSRQ) types and defines the parameters of the neural network, i.e. number of nodes in each layer and the leaky ReLU activation function parameter `alpha` (Leaky ReLU was selected forr all layers as it performs best).

```
"""
Script 2: Machine Learning Model
A neural network with one hidden layer is used to model signal reception properties
"""

import tensorflow as tf
import seaborn as sb
from random import randint

from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.layers import LeakyReLU
```

```
13 from tensorflow.keras.layers import Dense as Dense
14 from tensorflow.keras.models import Sequential
15
16 import matplotlib.pyplot as plt
17
18
19
20
21 """
22 Input position type definition (cartesian or spherical)
23 """
24
25 #input_pos_type = 'cartesian' #cartesian oordinates seem to perform better
26 input_pos_type = 'spherical'
27
28
29 """
30 Target type definition (RSRP or RSRQ)
31 """
32
33 target_type = 'RSRP'
34 #target_type = 'RSRQ'
35
36
37
38 """
39 Network property definition and data splitting
40 """
41
42 #number of nodes in the hidden layer
43 n_nodes_hl1 = 25
44 n_nodes_hl2 = 10
45 n_nodes_hl3 = 8
46 n_nodes_hl4 = 6
47 n_nodes_hl5 = 0
48
49 # alpha for leaky relu
50 alpha_lrelu = 0.1
```

The next step is the looped training. It starts with the train/test split, which uses the loop's index `ii` as seed. Then, the `norm` function (normalizing of data such that $\mu = 0$ and $\sigma^2 = 1$ improves machine learning performance) is defined and an optional statistical analysis may be put out. After the actual normalization

(lines 84 and 85), either RSRQ or RSRP values are popped from the normed training data, i.e. the actual target selection takes place.
The code block between lines 96 and 127 serves to choose either Cartesian or polar coordinates by deleting the other coordinate type from the data frame.
Then, the model is defined, starting with a declaration of the sequential architecture, which means that the hidden layers are simply put one after another without any recursion or any other advanced kind of architecture. Lines 139 through 157 define the hidden leaky ReLU layers. In the last layer, the previous outputs are simply summed up (line 159). After that, the actual training is started and the results are plotted and saved. Since this code segment is already quite lengthy and the plotting/saving plot is generic, it is left out for the most part. The only part that is relevant in the context of this thesis is the denormalization of the data before plotting.
Validation and test splits (i.e. the portion of data that is put into validation/test sets and thus not usable for learning) are set to very low values since the amount of data available is very small.

```
52  for ii in range(1000):
53      # split into test and training data
54      random_state = ii
55      train_dataset = df.sample(frac=0.99,random_state=random_state)
56      test_dataset = df.drop(train_dataset.index)
57
58
59
60      """
61      Function definitions
62      """
63
64      # calculate z-value -> normalize input data to [0,1]
65      def norm(x):
66        return (x - train_stats['mean']) / train_stats['std']
67
68
69
70      """
71      Data visualization - pre ML
72      """
73
74      #pairplot - comment to save time if not needed
75      #sb.pairplot(df,corner = True)
76
77
78
79      """
```

```
80      Machine Learning - Data preparation
81      """
82
83      #normalize data such that mu = 0 and sigma^2 = 1 and drop NaN-values
84      normed_train_data = norm(train_dataset).dropna()
85      normed_test_data = norm(test_dataset).dropna()
86
87
88      #split off target values
89      if target_type == 'RSRP':
90          train_labels = normed_train_data.pop('RSRP')
91          test_labels = normed_test_data.pop('RSRP')
92      if target_type == 'RSRQ':
93          train_labels = normed_train_data.pop('RSRQ')
94          test_labels = normed_test_data.pop('RSRQ')
95
96      #remove input values according to input type choice
97      if input_pos_type == 'cartesian':
98          if target_type == 'RSRQ':
99              del normed_train_data['RSRP']
100         if target_type == 'RSRP':
101             del normed_train_data['RSRQ']
102         del normed_train_data['azimuth']
103         del normed_train_data['elevation']
104         del normed_train_data['radius']
105         if target_type == 'RSRQ':
106             del normed_test_data['RSRP']
107         if target_type == 'RSRP':
108             del normed_test_data['RSRQ']
109         del normed_test_data['azimuth']
110         del normed_test_data['elevation']
111         del normed_test_data['radius']
112
113     if input_pos_type == 'spherical':
114         if target_type == 'RSRQ':
115             del normed_train_data['RSRP']
116         if target_type == 'RSRP':
117             del normed_train_data['RSRQ']
118         del normed_train_data['x']
119         del normed_train_data['y']
120         del normed_train_data['z']
```

```
121         if target_type == 'RSRQ':
122             del normed_test_data['RSRP']
123         if target_type == 'RSRP':
124             del normed_test_data['RSRQ']
125         del normed_test_data['x']
126         del normed_test_data['y']
127         del normed_test_data['z']
128 
129 
130 
131 
132     """
133     Machine Learning - Execution
134     """
135 
136 
137     # model for leaky relu activation function
138     model = keras.Sequential()
139     #first layer
140     model.add(Dense(n_nodes_hl1))
141     model.add(LeakyReLU(alpha=alpha_lrelu))
142     #second layer
143     if n_nodes_hl2 != 0:
144         model.add(Dense(n_nodes_hl2))
145         model.add(LeakyReLU(alpha=alpha_lrelu))
146     #third layer
147     if n_nodes_hl3 != 0:
148         model.add(Dense(n_nodes_hl3))
149         model.add(LeakyReLU(alpha=alpha_lrelu))
150     #fourth layer
151     if n_nodes_hl4 != 0:
152         model.add(Dense(n_nodes_hl4))
153         model.add(LeakyReLU(alpha=alpha_lrelu))
154     #fifth layer
155     if n_nodes_hl5 != 0:
156         model.add(Dense(n_nodes_hl5))
157         model.add(LeakyReLU(alpha=alpha_lrelu))
158     #addition layer
159     model.add(Dense(1))
160     model.compile(loss='mse',
161                   optimizer='adam',
```

```
162                     )
163
164         print("training started with random state "+np.str(random_state))
165
166         # train the model TAKE CARE FOR BATCH SIZE!
167         EPOCHS = 2000
168         history = model.fit( normed_train_data, train_labels, epochs=EPOCHS, \
169                             validation_split = 0.04, verbose=0, shuffle = True, \
170                             batch_size = 1024)
171
172         #summarize the model (inspect it)
173         model.summary()
```

After letting the script iterate up to `ii = 384`, the folder with the results was overlooked manually in search for a decent seed, which was found in number 127, see figure 51. This seed then serves as a starting point for another learning process that is not capped at 2000 epochs and runs until 10000 or 20000 epochs are completed. As a last remark, it has to be mentioned that these results are not exactly reproducible since the shuffling between epochs is done randomly, without a defined seed, see line 169. However, this approach works well for the problem at hand, as long as other parameters such as the batch size are kept constant. Deactivating shuffling leads to worse performance and modifying the code provided by the library would go too far at this point (although it would be possible if absolute reproducibility is needed).

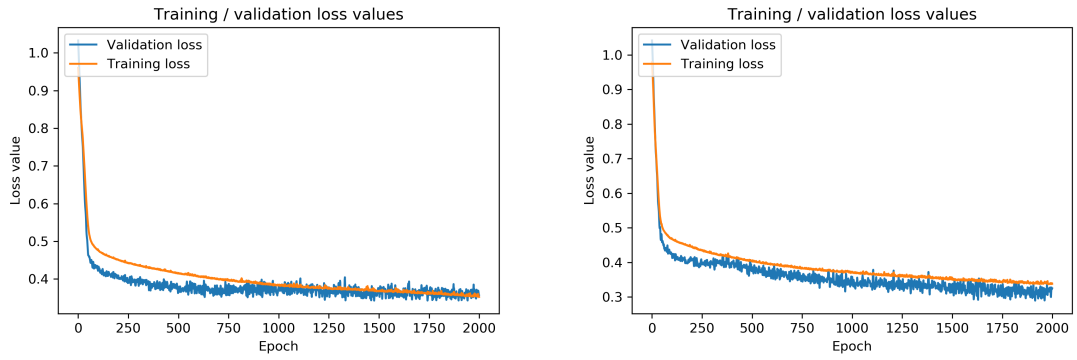


Figure 49: Training two times with the same seed shows that training behavior is reproducible for the most part.

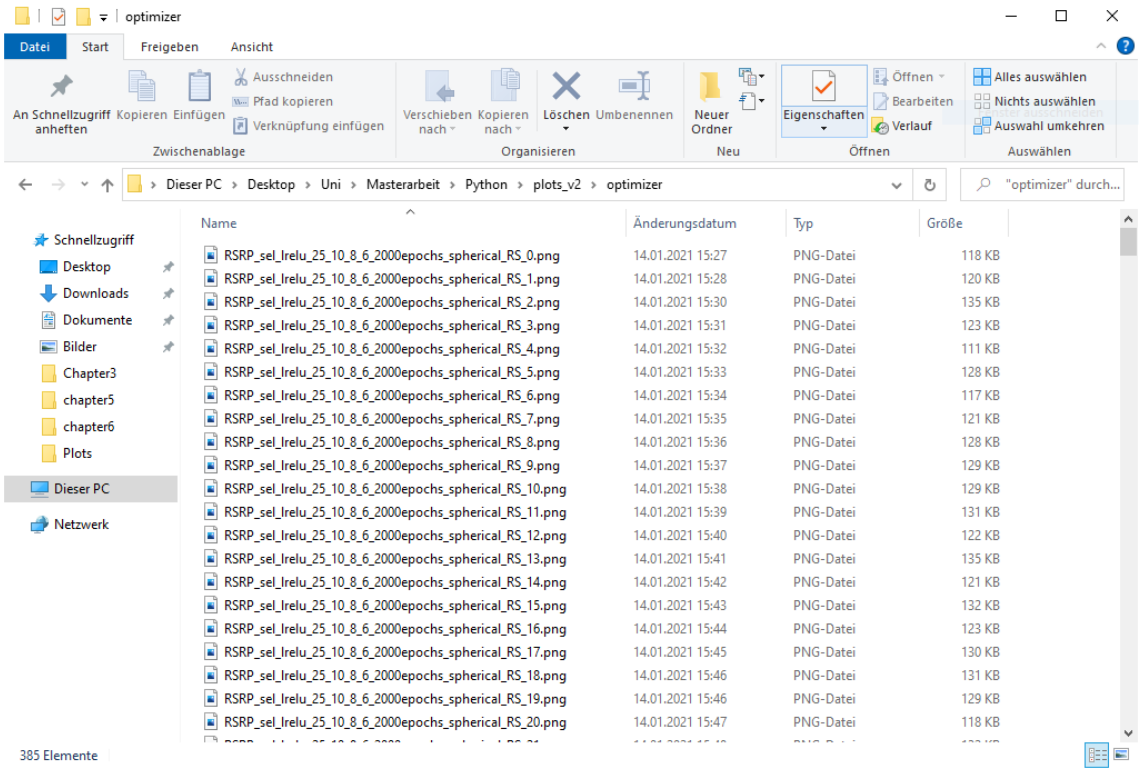


Figure 50: The folder with 385 learning curves

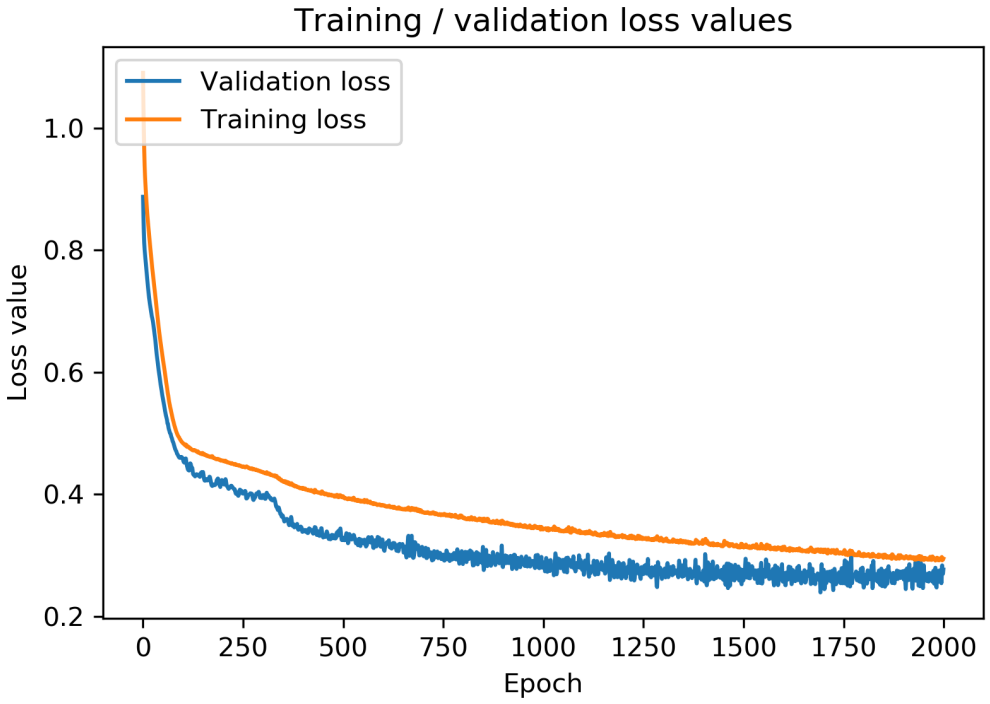


Figure 51: One example for a decently working seed (127).

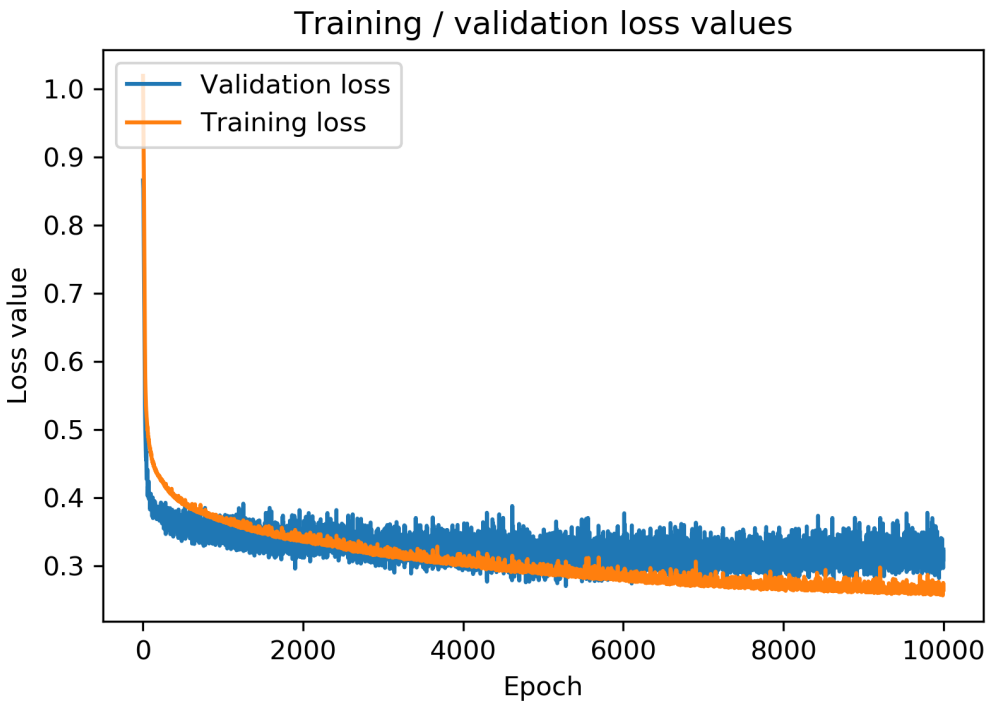


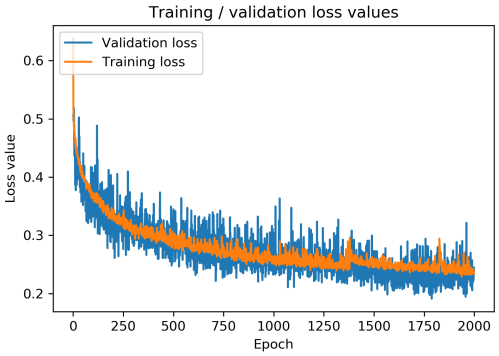
Figure 52: Training for 10000 epochs with seed 127. The neural net runs into overtraining after about 4000 epochs.

6.3 Choice of the batch size

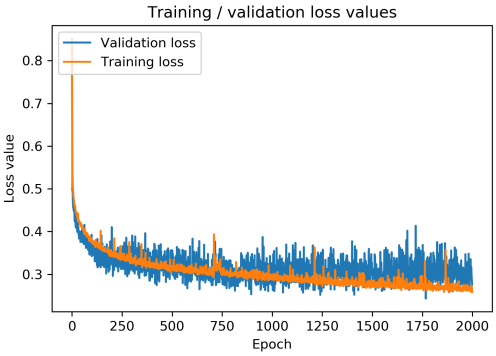
As discussed in chapter 4, the batch size influences learning behavior. In some instances, a smaller batch size may let the learning algorithm jump our of its trajectory towards a certain local minimum, which improves or worsens the results. In the context of this thesis, no such behavior occurred in a significant way[2] such that a batch size of 1024 was chosen. This value offered a good trade-off between learning duration, learning smoothness and resulting cost function. As for many aspects of machine learning, there is no scientifically optimal choice. Rather, one chooses a value that usually works decently and rolls with it.

As the figures 53(a) through 53(f) show, smaller batch sizes lead to a less smooth learning behavior, as one would expect. Furthermore, the training takes way more time as the amount of weight adaptations increases with decreasing batch sizes. This means that a comparison between 2000 epochs with a smaller batch size and the same amount of epochs with a larger batch size is actually not fair and a smaller cost function is to be expected for the smaller batches. They usually converge to the same value though.

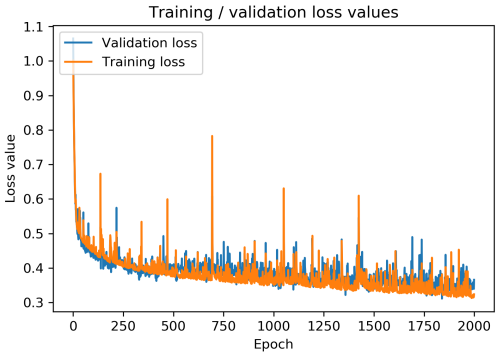
[2]It can be observed though, for example in figure 53(a), at around epoch 1400, where the training cost function suddenly worsens and the validation cost function suddenly improves. This hints at such a jump taking place.



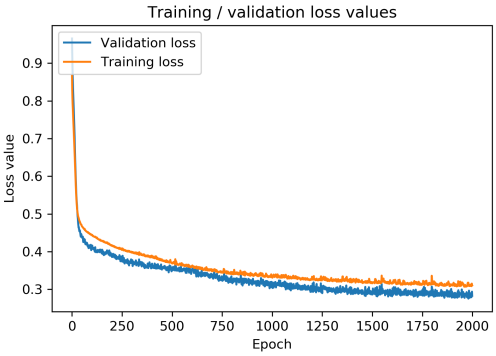
(a) Batch size = 1



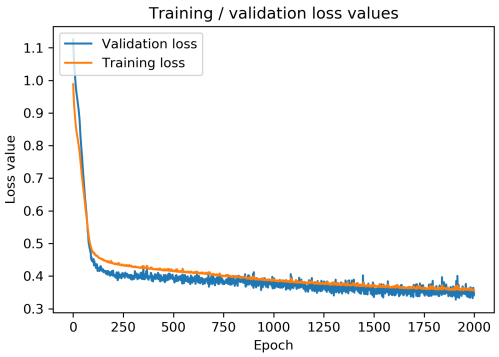
(b) Batch size = 16



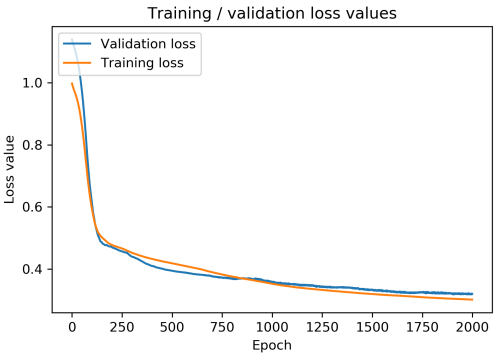
(c) Batch size = 128



(d) Batch size = 512



(e) Batch size = 1024



(f) Batch size = 8192

Figure 53: Training behavior for different batch sizes. Larger batch sizes tend to lead to a smoother learning behavior.

7 Discussion of results

This final chapter serves as a summary of the results and the findings of this thesis. Results for different flights are plotted and discussed, such that a starting point for further research is created.

7.1 Linear regression

The goal of the linear regression was to find a linear relationship between the distance between transmitter and receiver (the radius r) and the RSRP/RSRQ values on a log-log plot. In the following section, regression results are listed up and discussed.
It has to be noted that the test flights for linear regression were conducted in a relatively low frequency band, such that free space attenuation is lower than in more high frequency scenarios.

RSRP (Reference signal received power)

The following table contains regressions for all useful flights with their respective time, $RSRP(d_0)$ (i.e. RSRP at 100m) and the slope on the log-log plot k. All values are the weighted averages.

Flight number	Flight time	$RSRP(d_0)$	k
1	12:05:06	-58dBm	-26.76
2	12:26:56	-63dBm	-20.99
3	12:45:11	-61dBm	-22.82
4	12:05:06	-60dBm	-23.71

As already elaborated in chapter three, no model for these exact circumstance exists today. However, values for k are definitely in the plausible range, when compared to the predictions for similar circumstances, see terrain category C behavior in chapter 3. Furthermore, some kind of variation from theoretical predictions is to be expected because the actual path loss is not measured. Receiver and transmitter antenna patterns hugely influence reception and it can be assumed that multipath propagation has had a big influence on measurements. Actually, for many measurement points, a dominant line of sight component does presumably not even exist.
It can be concluded that even though the values found in this thesis show somewhat of a spread, they are still in a plausible range. Thus, linear regression remains a valid tool for modeling of large scale fading of UAV reception parameters.
As discussed in chapter three, it has to be noted though that higher elevation angles tend to lead to worse reception. Especially if the UAV is flown directly above a transmitter station, it is safe to assume that reception quality will suffer due to a stronger attenuation of non line of sight (NLOS) components that the user equipment receives. In reality, other transmitter stations will then establish the connection, but this aspect needs further research. Overall, the RSRP values were in a good range in all test flights though.

RSRQ (Reference signal received quality)

The following table shows the findings for RSRQ values, with d being the offset on the log-log plot, i.e. the RSRQ value at a distance of 100m:

Flight number	Flight time	d	k
1	12:05:06	-5.9	-5
2	12:26:56	-8.7	-2.9
3	12:45:11	-8	-3.9
4	12:05:06	-4.4	-6.9

For these values, no comparison to another model is possible. Furthermore, they strongly depend on influences that cannot be controlled, i.e. interference from other devices/transmitter stations. Since these disturbances may decrease with higher altitudes or higher distances from the transmitter, the slope may be positive. This is depicted in figure 54.

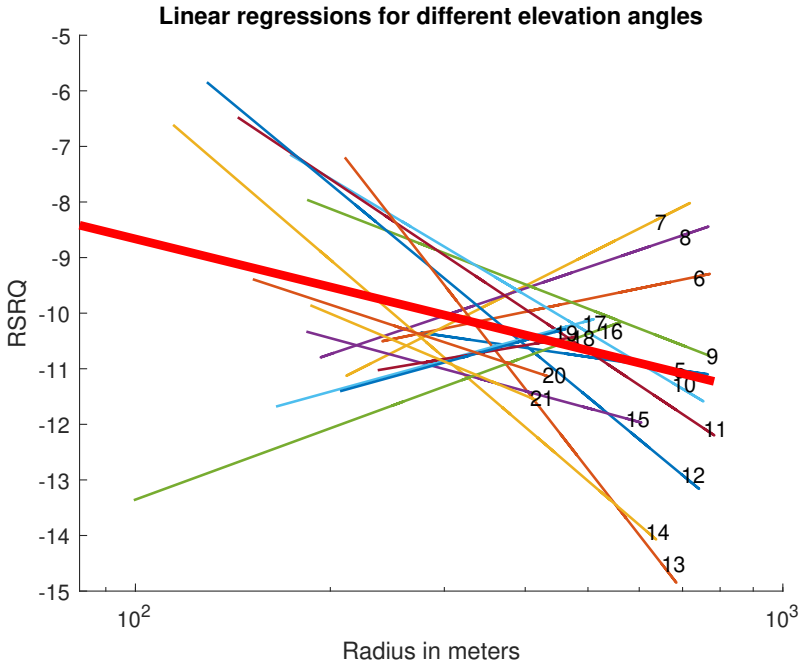


Figure 54: Calculated linear regressions for RSRQ and their weighted mean. As opposed to RSRP, the RSRQ value may increase with higher distances due to decreasing received interference.

7.2 Machine learning

Machine learning was used to build a neural network that is able to predict reception parameters for arbitrary points in space. The exact approach is discussed in chapters 4 and 6. As shall be shown, this approach works very decently as long as the UAV position, that RSRP/RSRQ should be predicted for, is not too far away from known measurement points.

Figures 55 and 56 show a comparison between measured data points and neural network predictions. They

look pretty similar, such that this results at least serves as a proof of concept. The larger circles in the estimated value graph are test data. They were not seen before by the neural network, such that they hint at the network's ability to generalize.
It has to be noted that only measurements from one transmitter station were taken into account for machine learning. Thus, a few data points are missing in figure 56.

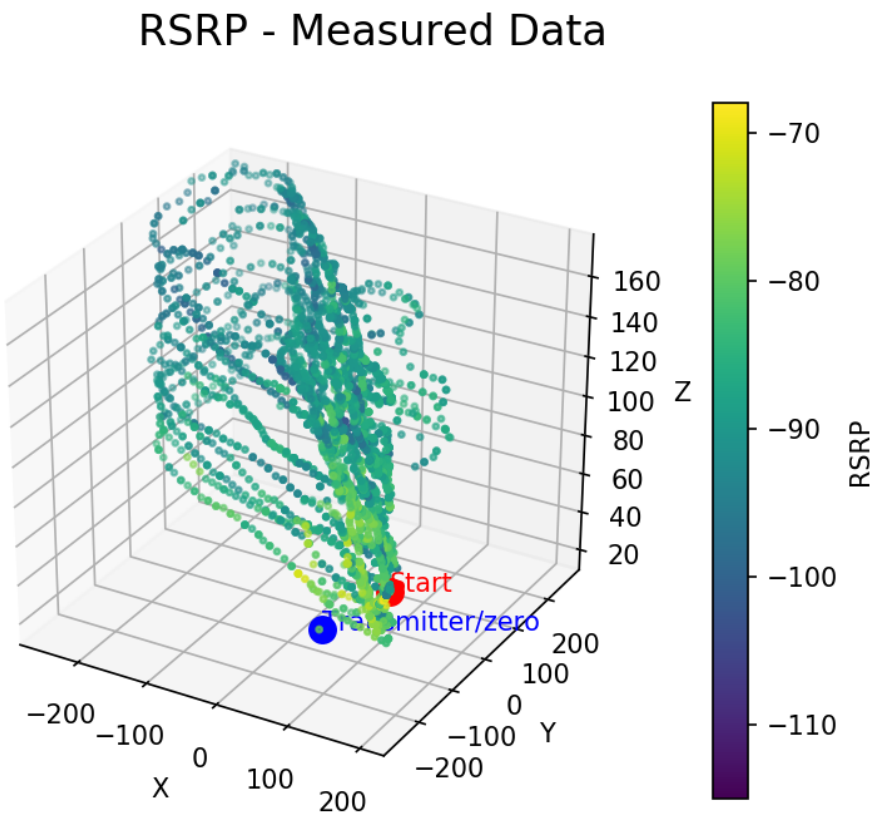


Figure 55: Machine learning measurement data. This dataset is the biggest one available and was thus used for all experiments as this approach strongly profits of higher amounts of data.

These results were achieved with a 25-10-8-6 network structure with leaky ReLU activation functions. This kind of deeper structures enable a somewhat intelligent prediction of RSRP/RSRQ values. For example, a neural network may implicitly take the transmitter signal's angle of arrival on the receiver into account. Of course, such assumptions are speculative in this case, but for computer vision applications, the inner workings are well documented, for example by [22].
Further testing was conducted by means of a plausibility check, see figure 57, which puts an equally spaced grid of positions into the network. Even though the network never puts out completely implausible values, it becomes obvious that higher distances from measurement points that the network was trained on lead to incorrect results. Thus, this approach is only valid in regions that are already known. Far extrapolations don't work. It has to be mentioned though that the used network is rather simple and with more effort and more training data (from a wider variety of places/scenarios), a proper prediction of reception parameters may be built.

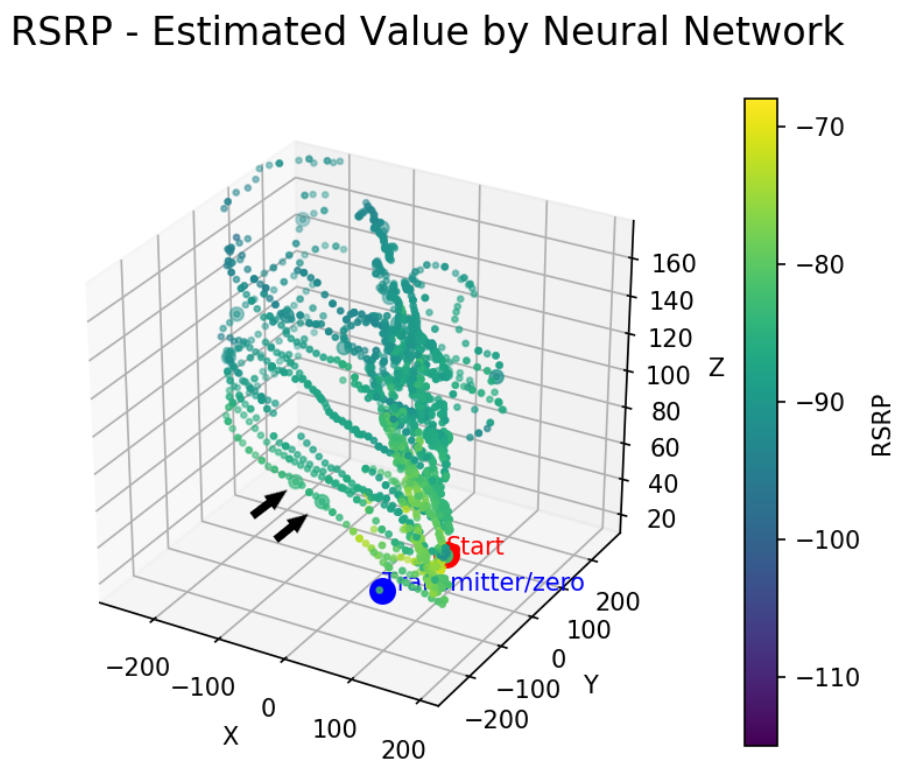


Figure 56: Predicted values by the trained neural network. Bigger circles are test data which the network has not seen before. Two of these data points are marked with arrows.

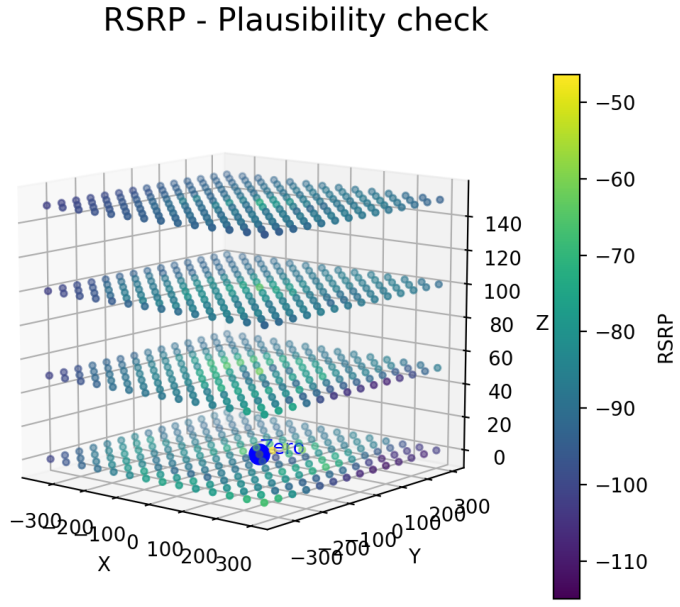


Figure 57: Plausibility check. The kind of neural network used works well for nonlinear regression in regions close to measurement points. Bigger distances lead to worse results/prediction failure.

8 Conclusions and outlook

For both used techniques, linear regression and neural networks, this thesis is at least a proof of concept. They both work for the task at hand, but the high complexity of this field means that there are a few caveats. Linear regression cannot be applied in a straightforward way. Rather, the effect of the elevation angle between base station and user equipment must be taken into account. The proposed functions to model this interconnection serve as a starting point for others, who can develop more precise models, based on these discoveries. It has to be determined, which functions are useful for this kind of task and how to fit them properly.

Artificial neural networks have been shown to be capable of predicting cellular reception quality parameters as long as proximity to known (i.e. learned) measurement points is given. Even relatively simple feedforward neural networks, which simply take UAV coordinates as input suffice to accomplish this task. Future development could for example lead to early warning systems, which predict dead zones in a UAVs trajectory. To develop such systems, more data from many different scenarios will be needed. Furthermore, it makes sense to add other input variables like population density in the area of operation and construct more complex artificial intelligence, which is better suited for such critical tasks.

With 5G (and thus massive MIMO and beamforming) around the corner, investigations like the thesis at hand need to be repeated for this new technology. Especially for steered beams, linear regression may even be more useful for cellular reception quality parameter prediction. However, the higher frequencies will lead to many new challenges, especially due to the higher free space attenuation. The shorter wavelengths may also mean that the validity of the predicted 5G reception quality parameters (RSRP/RSRQ are relevant only for 4G) values by neural networks are confined to smaller spaces around known points.

Bibliography

[1] Telesystem Innovations. *LTE in a Nutshell: The Physical Layer*. Accessed on 2nd of March, 2021, from: https://home.zhaw.ch/kunr/NTM1/literatur/LTE%20in%20a%20Nutshell%20-%20Physical%20Layer.pdf

[2] CableFree. *RSRP and RSRQ Measurement in LTE*. Accessed on 2nd of March, 2021, from: https://www.cablefree.net/wirelesstechnology/4glte/rsrp-rsrq-measurement-lte/

[3] LTE-Anbieter.info. *RSRP (Referenz[sic!] Signal Received Power)*. Accessed on 2nd of March, 2021, from: https://www.lte-anbieter.info/technik/rsrp.php

[4] LTE-Anbieter.info. *RSSI (Received Signal Strength Indicator)*. Accessed on 15th of February, 2021, from: https://www.lte-anbieter.info/technik/rssi.php

[5] LTE-Anbieter.info. *RSRQ (Reference Signal Received Quality)*. Accessed on 15th of February, 2021, from: https://www.lte-anbieter.info/technik/rsrq.php

[6] jusline.at. *$ 24c LFG(Luftfahrtgesetz).* Accessed on 22nd of February 2021, from: https://www.jusline.at/gesetz/lfg/paragraf/24c

[7] dronerules.eu. *EU Regulations Updates.* Accessed on 23rd of February 2021, from: https://dronerules.eu/sl/professional/eu_regulations_updates

[8] dronespace.at. *"Specific" Kategorie.* Accessed on 23rd of February 2021, from: https://dronespace.at/specific

[9] V. Erceg, L. J. Greenstein, S. Y. Tjandra, S. R. Parkoff, A. Gupta, B. Kulic, A. A. Julius, and R. Bianchi. *An empirically based path loss model for wireless channels in suburban environments*. IEEE J. Select. Areas Commun., vol. 17, pp. 1205–1211, July 1999

[10] XIRIO ONLINE. *STANFORD UNIVERSITY INTERIM* Accessed on 11th of January, 2021, from: https://www.xirio-online.com/web/help/en/sui.html

[11] Hsing-Yi Chen and Tsung-Han Lin. *Simulations and Measurements of Electric Fields Emitted from a LTE Base Station in an Urban Area*. 2014, Department of Communications Engineering, Yuan Ze University, 135 Yuan-Tung Road, Nei-Li, Chung-Li, Taoyuan Shian 320, Taiwan

[12] Theodore S. Rappaport. *Wireless Communications, Principles and Practice*. 2002, 1996 Prentice Hall, Inc. ISBN: 0-13-0422332-0

[13] Ali Grami *Scale Fading*. Introduction to Digital Communications, 2016 Accessed on 11th of January, 2021, from: https://www.sciencedirect.com/topics/engineering/scale-fading

[14] Yong Soo Cho, Jaekwon Kim, Won Young Yang and Chung G. Kang *MIMO-OFDM Wireless Communications with MATLAB ®*. 2010 John Wiley & Sons (Asia) Pte Ltd. ISBN: 978-0-470-82561-7 Accessed on 11th of January, 2021, from: http://read.pudn.com/downloads566/ebook/2330540/The%20Wireless%20Channel%20Propagation.pdf

[15] Cory Maklin. *LSTM Recurrent Neural Network Keras Example*. Accessed on 14th of December, 2020, from: https://towardsdatascience.com/machine-learning-recurrent-neural-networks-and-long-short-term-memory-lstm-python-keras-example-86001ceaaebc, June 2019

[16] Wikipedia-User Chrislb. Accessed on 14th of December, 2020, from: https://commons.wikimedia.org/wiki/File:ArtificialNeuronModel_deutsch.png#/media/Datei:ArtificialNeuronModel_deutsch.png, July 2005

[17] Xavier Glorot, Antoine Bordes, Yoshua Bengio. *Deep Sparse Rectifier Neural Networks.* Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (= Proceedings of Machine Learning Research). Band 15. PMLR, Fort Lauderdale, FL, USA April 2011, P. 315–323 (mlr.press [PDF]).

[18] Shubham Deshmukh. *Dying ReLU Problem.* Accessed on 8th of March 2021, from: https://medium.com/@shubham.deshmukh705/dying-relu-problem-879cec7a687f

[19] Stylianos (Stelios) Kampakis. *What deep learning is and isn't.* Accessed on 17th of December, 2020, from: https://thedatascientist.com/what-deep-learning-is-and-isnt/, April 2018.

[20] Imad Dabbura. *Gradient Descent Algorithm and Its Variants.* Accessed on 13th of January, 2021 from: https://towardsdatascience.com/gradient-descent-algorithm-and-its-variants-10f652806a3/, Dec 2017

[21] Hind Sellouk. *Matrix Based Back-propagation.* Accessed on 08th of February, 2021 from: https://medium.com/@hindsellouk13/matrix-based-back-propagation-fe143ce2b2df, Jul 2018

[22] Olah, et al. *The Building Blocks of Interpretability.* Distill, 2018. Accessed on 1st of march, 2021 from: https://distill.pub/2018/building-blocks/

Appendix A - MATLAB scripts

Analysis.m

```
clc
clear all
close all

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Data from: 865116045125976_8_10_2020_12.5.6.azm.csv
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%               ANALYSIS PARAMETERS
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Select "Corridor" between elevation_min and elevation_max
elevation_min = 10;
elevation_max = 11;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Select "Corridor" between azimuth_min and azimuth_max
% Better not change these, once properly selected
azimuth_min = 73;
azimuth_max = 76;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%



%% Import data from text file.
% Script for importing data from the following text file:
%
%    C:\Users\Gianni\Desktop\Uni\Masterarbeit\Neue ...
    Fluege\DirnbachConverted\CSV\865116045125976_8_10_2020_12.5.6.azm.csv
%
% To extend the code to different selected data or a different text file,
% generate a function instead of a script.

% Auto-generated by MATLAB on 2020/11/07 14:46:54

%% Initialize variables.
filename = '.\DirnbachConverted\CSV\865116045125976_8_10_2020_12.5.6.azm.csv';
delimiter = ',';
startRow = 2;

%% Format for each line of text:
```

```
%   column1: text (%q)
%   column2: double (%f)
%   column3: double (%f)
%   column4: double (%f)
%   column5: double (%f)
%   column6: double (%f)
%   column7: double (%f)
%   column8: double (%f)
%   column9: double (%f)
%   column10: double (%f)
%   column11: double (%f)
%   column12: double (%f)
%   column13: double (%f)
%   column14: double (%f)
%   column15: double (%f)
%   column16: double (%f)
%   column17: double (%f)
%   column18: double (%f)
%   column19: double (%f)
%   column20: double (%f)
%   column21: double (%f)
%   column22: double (%f)
% For more information, see the TEXTSCAN documentation.
formatSpec = '%q%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%[^\n\r]';

%% Open the text file.
fileID = fopen(filename,'r');

%% Read columns of data according to the format.
% This call is based on the structure of the file used to generate this
% code. If an error occurs for a different file, try regenerating the code
% from the Import Tool.
dataArray = textscan(fileID, formatSpec, 'Delimiter', delimiter, 'TextType', ...
    'string', 'EmptyValue', NaN, 'HeaderLines' ,startRow-1, 'ReturnOnError', ...
    false, 'EndOfLine', '\r\n');

%% Close the text file.
fclose(fileID);

%% Post processing for unimportable data.
% No unimportable data rules were applied during the import, so no post
% processing code is included. To generate code which works for
% unimportable data, select unimportable cells in a file and regenerate the
% script.

%% Create output variable
Dataframe = table(dataArray{1:end-1}, 'VariableNames', ...
    {'time','longitude','latitude','altitude','height','rsrp1','sinr1','pci1','cgi', ...
    ...
    'rsrq1','pci2','rsrp2','sinr2','rsrq2','pci3','rsrp3','sinr3','rsrq3','pci4', ...
    ,'rsrp4','sinr4','rsrq4'});

```

```
%% Clear temporary variables
clearvars filename delimiter startRow formatSpec fileID dataArray ans;



%% Start of actual analysis script
% by Giancarlo Benincasa
% 11/2020
% Data from: 865116045125976_8_10_2020_12.5.6.azm.csv

% get RSRP and RSRQ
RSRP = Dataframe.rsrp1;
RSRQ = Dataframe.rsrq1;

% get postitional information (geocoordinates/height above sea level)
altitude = Dataframe.altitude;
latitude = Dataframe.latitude;
longitude = Dataframe.longitude;

%Transmitter Station coordinates
LongitudeTx = 15.892169;
LatitudeTx = 46.830863;

% convert them to x,y,z
x = (longitude - LongitudeTx)*76000;
y = (latitude - LatitudeTx)*111000;
z = altitude-min(altitude);

% Coordinate transforms
[azimuth,elevation,r] = cart2sph(x,y,z);
azimuth = azimuth*180/pi;
elevation = elevation*180/pi;

% Polar coordinate matrix
X = [azimuth,elevation,z]';


%% Path loss model

% set target (RSRP or RSRQ)
t = RSRP;



% 3-D-Flightplot with RSRP
fontsize = 18;
figure
hold on
%Route
plot3(x,y,z)
%Measurements
```

```
size = 50;
color = t;
scatter3(x,y,z,size,color,'filled')
ax = gca;
ax.FontSize = fontsize;
c = colorbar;
c.Label.String = '\fontsize{18} RSRP';

%Start
size = 150;
scatter3(x(1),y(1),z(1),size,'red','filled')
%Transmitter position
size = 200;
scatter3(0,0,0,size,'blue','filled')
title('Flight route with RSRP; Measurement data',"FontSize",fontsize)
legend('\fontsize{18} Interpolation (linear)','\fontsize{18} ...
    Measurements','\fontsize{18} Start','\fontsize{18} Transmitter station')
caxis([min(t) max(t)])

% Plot a histogram of angles to get values of phi and theta to analyze
figure
subplot(2,1,1)
histogram(azimuth)
xlabel("azimuth")
subplot(2,1,2)
histogram(elevation)
xlabel("elevation")
suptitle("Histogram of angles")

% throw away RSRP/RSRQ data that stems from > azimuth_max
t_sel = t;
azimuth_sel = azimuth;
elevation_sel = elevation;
r_sel = r;
x_sel = x;
y_sel = y;
z_sel = z;
t_sel(azimuth > azimuth_max) = [];
azimuth_sel(azimuth > azimuth_max) = [];
elevation_sel(azimuth > azimuth_max) = [];
r_sel(azimuth > azimuth_max) = [];
x_sel(azimuth > azimuth_max) = [];
y_sel(azimuth > azimuth_max) = [];
z_sel(azimuth > azimuth_max) = [];

% throw away < azimuth_min
t_sel(azimuth_sel < azimuth_min) = [];
elevation_sel(azimuth_sel < azimuth_min) = [];
r_sel(azimuth_sel < azimuth_min) = [];
x_sel(azimuth_sel < azimuth_min) = [];
y_sel(azimuth_sel < azimuth_min) = [];
z_sel(azimuth_sel < azimuth_min) = [];
```

```
azimuth_sel(azimuth_sel < azimuth_min) = [];


% throw away > elevation_max
t_sel(elevation_sel > elevation_max) = [];
azimuth_sel(elevation_sel > elevation_max) = [];
r_sel(elevation_sel > elevation_max) = [];
x_sel(elevation_sel > elevation_max) = [];
y_sel(elevation_sel > elevation_max) = [];
z_sel(elevation_sel > elevation_max) = [];
elevation_sel(elevation_sel > elevation_max) = [];

% throw away < elevation_min
t_sel(elevation_sel < elevation_min) = [];
azimuth_sel(elevation_sel < elevation_min) = [];
r_sel(elevation_sel < elevation_min) = [];
x_sel(elevation_sel < elevation_min) = [];
y_sel(elevation_sel < elevation_min) = [];
z_sel(elevation_sel < elevation_min) = [];
elevation_sel(elevation_sel < elevation_min) = [];


% 3D plot selected flight data
% (change labels/titles manually to RSRQ if needed)
figure
hold on
%Measurements
size = 50;
color = t_sel;
scatter3(x_sel,y_sel,z_sel,size,color,'filled')
c = colorbar;
c.Label.String = 'RSRP';
%Start
size = 150;
scatter3(x(1),y(1),z(1),size,'red','filled')
%Transmitter position
size = 200;
scatter3(0,0,0,size,'blue','filled')
title('Selected RSRP values from flight route; Measurement data')
legend('Measurements','Start','Transmitter station')
caxis([min(t_sel) max(t_sel)])

% TEST OF LOGLOG MODEL
r_sel_log = log10(r_sel/100);           % d0 = 100m

% calculate linear regression
kd_mat = [ones(length(r_sel_log),1) r_sel_log];
kd = kd_mat\t_sel;
t_linreg = kd_mat*kd;


% Plot relationship between radius and RSRP, only plot linear regression
```

```
% (change labels/titles manually to RSRQ if needed)
figure
hold on
scatter(100*10.^(r_sel_log),t_sel)
plot(100*10.^(r_sel_log),t_linreg,"LineWidth",2)
set(gca,'xscale','log')
legend("Measured data","Linear regression")
xlabel("radius in meters")
ylabel("RSRP in dBm")
grid on
title("Relationship between radius and RSRP, \theta in range (" + ...
    num2str(elevation_min) + "degrees," + num2str(elevation_max) +"degrees )")
```

Automated_Analysis.m

```
clc
clear all
close all

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Data from: 865116045125976_8_10_2020_12.5.6.azm.csv
% can be changed though
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%       USE THIS SCRIPT TO ANALYZE RSRP FOR
%       MULTIPLE ELEVATION ANGLES AT ONCE
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%               ANALYSIS PARAMETERS
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Select "Corridors" between elevation_min and elevation_max
elevation_min = 4:20;
elevation_max = 5:21;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Select "Corridor" between azimuth_min and azimuth_max
% Better not change these, once properly selected
azimuth_min = 59;
azimuth_max = 61;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% set target (RSRP or RSRQ)
target = "RSRP"




%% Import data from text file.
```

```
% Script for importing data from the following text file:
%
%    C:\Users\Gianni\Desktop\Uni\Masterarbeit\Neue ...
    Fluege\DirnbachConverted\CSV\865116045125976_8_10_2020_12.5.6.azm.csv
%
% To extend the code to different selected data or a different text file,
% generate a function instead of a script.

% Auto-generated by MATLAB on 2020/11/07 14:46:54

%% Initialize variables.
%filename = '.\DirnbachConverted\CSV\865116045125976_8_10_2020_12.5.6.azm.csv';
%filename = '.\DirnbachConverted\CSV\865116045125976_8_10_2020_12.26.56.azm.csv';
filename = '.\DirnbachConverted\CSV\865116045125976_8_10_2020_12.45.11.azm.csv';
%filename = '.\DirnbachConverted\CSV\865116045125976_8_10_2020_13.58.28.azm.csv';
delimiter = ',';
startRow = 2;

%% Format for each line of text:
%   column1: text (%q)
%   column2: double (%f)
%   column3: double (%f)
%   column4: double (%f)
%   column5: double (%f)
%   column6: double (%f)
%   column7: double (%f)
%   column8: double (%f)
%   column9: double (%f)
%   column10: double (%f)
%   column11: double (%f)
%   column12: double (%f)
%   column13: double (%f)
%   column14: double (%f)
%   column15: double (%f)
%   column16: double (%f)
%   column17: double (%f)
%   column18: double (%f)
%   column19: double (%f)
%   column20: double (%f)
%   column21: double (%f)
%   column22: double (%f)
% For more information, see the TEXTSCAN documentation.
formatSpec = '%q%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%[^\n\r]';

%% Open the text file.
fileID = fopen(filename,'r');

%% Read columns of data according to the format.
% This call is based on the structure of the file used to generate this
% code. If an error occurs for a different file, try regenerating the code
% from the Import Tool.
```

```
85  dataArray = textscan(fileID, formatSpec, 'Delimiter', delimiter, 'TextType', ...
        'string', 'EmptyValue', NaN, 'HeaderLines' ,startRow-1, 'ReturnOnError', ...
        false, 'EndOfLine', '\r\n');
86
87  %% Close the text file.
88  fclose(fileID);
89
90  %% Post processing for unimportable data.
91  % No unimportable data rules were applied during the import, so no post
92  % processing code is included. To generate code which works for
93  % unimportable data, select unimportable cells in a file and regenerate the
94  % script.
95
96  %% Create output variable
97  Dataframe = table(dataArray{1:end-1}, 'VariableNames', ...
        {'time','longitude','latitude','altitude','height','rsrp1','sinr1','pci1','cgi', ...
        ...
98      'rsrq1','pci2','rsrp2','sinr2','rsrq2','pci3','rsrp3','sinr3','rsrq3','pci4', ...
99      ,'rsrp4','sinr4','rsrq4'});
100
101 %% Clear temporary variables
102 clearvars filename delimiter startRow formatSpec fileID dataArray ans;
103
104
105
106
107 %% Start of actual analysis script
108 % by Giancarlo Benincasa
109 % 11/2020
110 % Data from: 865116045125976_8_10_2020_-.-.-.azm.csv
111
112 % get RSRP
113 RSRP = Dataframe.rsrp1;
114 RSRQ = Dataframe.rsrq1;
115
116 % set target
117 if target == "RSRP"
118     t = RSRP;
119 elseif target == "RSRQ"
120     t = RSRQ;
121 else
122     fprintf("Target must be RSRP or RSRQ. Defaulted to RSRP.")
123     t = RSRP
124 end
125
126 % get postitional information (geocoordinates/height above sea level)
127 altitude = Dataframe.altitude;
128 latitude = Dataframe.latitude;
129 longitude = Dataframe.longitude;
130
131 %Transmitter Station coordinates
132 LongitudeTx = 15.892169;
```

```
LatitudeTx = 46.830863;

% convert them to x,y,z
x = (longitude - LongitudeTx)*76000;
y = (latitude - LatitudeTx)*111000;
z = altitude-min(altitude);

% Coordinate transforms
[azimuth,elevation,r] = cart2sph(x,y,z);
azimuth = azimuth*180/pi;
elevation = elevation*180/pi;

% Polar coordinate matrix
X = [azimuth,elevation,z]';


%% Path loss model


% 3-D-Flightplot with RSRP
figure
hold on
%Route
plot3(x,y,z)
%Measurements
size = 50;
color = t;
scatter3(x,y,z,size,color,'filled')
c = colorbar;
c.Label.String = 'RSRP';
%Start
size = 150;
scatter3(x(1),y(1),z(1),size,'red','filled')
%Transmitter position
size = 200;
scatter3(0,0,0,size,'blue','filled')
title('Flight route with RSRP; Measurement data')
legend('Interpolation (linear)','Measurements','Start','Transmitter station')
caxis([min(t) max(t)])

% Plot a histogram of angles to get values of phi and theta to analyze
figure
subplot(2,1,1)
histogram(azimuth,(1:2:100))
xlabel("azimuth")
subplot(2,1,2)
histogram(elevation,(1:2:100))
xlabel("elevation")
title("Histogram of angles")

% throw away RSRP data that stems from > azimuth_max
t_sel = t;
```

```
azimuth_sel = azimuth;
elevation_sel = elevation;
r_sel = r;
x_sel = x;
y_sel = y;
z_sel = z;
t_sel(azimuth > azimuth_max) = [];
azimuth_sel(azimuth > azimuth_max) = [];
elevation_sel(azimuth > azimuth_max) = [];
r_sel(azimuth > azimuth_max) = [];
x_sel(azimuth > azimuth_max) = [];
y_sel(azimuth > azimuth_max) = [];
z_sel(azimuth > azimuth_max) = [];

% throw away < azimuth_min
t_sel(azimuth_sel < azimuth_min) = [];
elevation_sel(azimuth_sel < azimuth_min) = [];
r_sel(azimuth_sel < azimuth_min) = [];
x_sel(azimuth_sel < azimuth_min) = [];
y_sel(azimuth_sel < azimuth_min) = [];
z_sel(azimuth_sel < azimuth_min) = [];
azimuth_sel(azimuth_sel < azimuth_min) = [];

% save values to repeat analysis later
t_cleaned = t_sel;
elevation_cleaned = elevation_sel;
r_cleaned = r_sel;
x_cleaned = x_sel;
y_cleaned = y_sel;
z_cleaned = z_sel;
azimuth_cleaned = azimuth_sel;

% initialize matrices to save results in
kd_saved = [];
beta_saved = [];
angles_saved = [];
nr_datapoints_saved = [];

%% ANALYZE REGRESSION PARAMETERS

% This figure will be used to plot linreg results
figure(3)
hold on
title("Linear regressions for different elevation angles")
xlabel("Radius in meters")
if target == "RSRQ"
    ylabel("RSRQ")
elseif target == "RSRP"
    ylabel("RSRP in dBW")
end

% TAKE LOG OF POSITIONAL VARIABLES
```

```
% r_cleaned = log(r_cleaned);
% x_cleaned = log(x_cleaned);
% y_cleaned = log(y_cleaned);
% z_cleaned = log(z_cleaned);
% r = log(r);

for ii = 1:1:length(elevation_max)

    t_sel = t_cleaned;
    azimuth_sel = azimuth_cleaned;
    elevation_sel = elevation_cleaned;
    r_sel = r_cleaned;
    x_sel = x_cleaned;
    y_sel = y_cleaned;
    z_sel = z_cleaned;

    % throw away > elevation_max
    t_sel(elevation_sel > elevation_max(ii)) = [];
    azimuth_sel(elevation_sel > elevation_max(ii)) = [];
    r_sel(elevation_sel > elevation_max(ii)) = [];
    x_sel(elevation_sel > elevation_max(ii)) = [];
    y_sel(elevation_sel > elevation_max(ii)) = [];
    z_sel(elevation_sel > elevation_max(ii)) = [];
    elevation_sel(elevation_sel > elevation_max(ii)) = [];

    % throw away < elevation_min
    t_sel(elevation_sel < elevation_min(ii)) = [];
    azimuth_sel(elevation_sel < elevation_min(ii)) = [];
    r_sel(elevation_sel < elevation_min(ii)) = [];
    x_sel(elevation_sel < elevation_min(ii)) = [];
    y_sel(elevation_sel < elevation_min(ii)) = [];
    z_sel(elevation_sel < elevation_min(ii)) = [];
    elevation_sel(elevation_sel < elevation_min(ii)) = [];

    % LOGLOG MODEL
    r_sel = log10(r_sel/100);

    % calculate linear regression
    kd_mat = [ones(length(r_sel),1) r_sel];
    kd = kd_mat\t_sel;
    RSRP_linreg = kd_mat*kd;

    % save data
    kd_saved = [kd_saved kd];
    angles_saved = [angles_saved (elevation_min(ii)+elevation_max(ii))/2];
    nr_datapoints_saved = [nr_datapoints_saved length(r_sel)];

    % plot regression and elevation angle
    plot(100*10.^(r_sel),(kd(1)+r_sel*kd(2)),"LineWidth",1.2)
    txt = num2str(elevation_max(ii));
    text(100*10.^(r_sel(1)),(kd(1)+r_sel(1)*kd(2)),txt)

```

```
289 end
290
291
292 % calculate mean k and d values, weighted by number of datapoints
293 kmean = sum(kd_saved(2,:).*nr_datapoints_saved(:).')/sum(nr_datapoints_saved);
294 dmean = sum(kd_saved(1,:).*nr_datapoints_saved(:).')/sum(nr_datapoints_saved);
295 %plot weighted mean regression line
296 plot((min(r):max(r)),(dmean+log10((min(r):max(r))/100)*kmean),"LineWidth",4,"Color","red")
297 txt = "weighted mean";
298 text(123,-60,txt)
299 set(gca,'xscale','log')
300
301 %% Plot of results
302 title_FS = 22;  %font size
303 figure
304 subplot(2,1,1)
305 hold on
306 yyaxis left
307 plot(angles_saved, kd_saved(1,:),"LineWidth",2)
308 plot(angles_saved,ones(1,length(angles_saved))*dmean)
309 yyaxis right
310 plot(angles_saved, kd_saved(2,:),"LineWidth",2)
311 xlabel("elevation angle","FontSize",title_FS-2)
312 plot(angles_saved,ones(1,length(angles_saved))*kmean)
313 grid on
314 lgd = legend("d in dBm","weighted mean of d","k (log-log slope)","weighted mean ...
        of k");
315 lgd.FontSize =18;
316 title("linear equation coefficients as a function of elevation ...
        angle","FontSize",title_FS)
317 subplot(2,1,2)
318 plot(angles_saved,nr_datapoints_saved,"LineWidth",2)
319 title("Number of datapoints","FontSize",title_FS)
320 xlabel("elevation angle","FontSize",title_FS-2)
321 ylabel("# datapoints","FontSize",title_FS-2)
322 grid on
```

Appendix B - Jupyter Notebooks

MA_v2.ipynb

Reading and preprocessing of data

```
"""
Script 1: Reading and preprocessing of data from txt files

TO LOOK UP CELLS: go to http://www.opencellid.org
                  search for pirching an der raab
"""

import pandas
import numpy as np
import math









"""
Function definitions
"""

def cart2sph(x, y, z):
    hxy = np.hypot(x, y)
    r = np.hypot(hxy, z)
    el = np.arctan2(z, hxy)
    az = np.arctan2(y, x)
    return az, el, r




"""
Get measurement data from csv file
"""

#Measurement dataframe
meas_df = pandas.read_csv('Messdaten3.csv', sep=';')

#Connection values, use float to enable NaN
```

```
#throw away first values later!
RSRP = pandas.Series(meas_df.Level,dtype='float32')
RSRQ = pandas.Series(meas_df.Qual,dtype='float32')



"""
Process measurement data and prepare it for further usage
"""

#Position Values
LongitudeMobile = meas_df.Longitude
LatitudeMobile = meas_df.Latitude
HeightMobile = meas_df.Height

#get rid of excess dots
LongitudeMobileArray = np.zeros(len(LongitudeMobile)-1)
for i in range(len(LongitudeMobile)-1):
    test = LongitudeMobile[i+1]
    newstr = test[:6] + test[7:]
    LongitudeMobileArray[i] = float(newstr)
LatitudeMobileArray = np.zeros(len(LatitudeMobile)-1)
for i in range(len(LatitudeMobile)-1):
    test = LatitudeMobile[i+1]
    newstr = test[:6] + test[7:]
    LatitudeMobileArray[i] = float(newstr)

#preprocess height
HeightMobileArray = np.zeros(len(HeightMobile)-1)
minheight = min(HeightMobile[1:]) # throw away first three values
for i in range(1,len(HeightMobile)):
    h = HeightMobile[i]
    HeightMobileArray[i-1] = float(h-minheight)
z = HeightMobileArray

#Conversion from coordinates to meters - mean is chosen as zero
x = (LongitudeMobileArray - np.mean(LongitudeMobileArray))*76000;
y = (LatitudeMobileArray - np.mean(LatitudeMobileArray))*111000;
#Factors obtained from website:
#https://rechneronline.de/geo-koordinaten/#entfernung

```

```
78
79  # create dict of raw cell IDs
80  cellIDs = meas_df.RAWCELLID
81  IDnumber = 0 #number that is assigned to each unique cell ID, starting with zero
82  IDdict = {} #dictionary containing cell IDs and their respective numbers
83  for i in range(len(cellIDs)):
84      if not cellIDs[i] in IDdict:
85          IDdict[cellIDs[i]] = IDnumber
86          IDnumber = IDnumber + 1
87  IDnumberMax = IDnumber
88  print("ID dictionary:")
89  print(IDdict)
90
91  # create array of ID numbers
92  #IDnumberArray = np.zeros(len(cellIDs))    # np array
93  IDnumberArray = []                         # python array
94  for i in range(len(cellIDs)):
95      #IDnumberArray[i] = int(IDdict[cellIDs[i]])      # np array
96      IDnumberArray.append(int(IDdict[cellIDs[i]]))    # python array
97
98  # check for number of station appearances
99  stationCount = 0
100 for j in range(IDnumberMax):
101     for i in range(len(cellIDs)):
102         if IDnumberArray[i] == j:
103             stationCount = stationCount + 1
104     print("IDnumber "+str(j)+" appears "+str(stationCount)+" times.")
105     stationCount = 0
106
107
108 # Replace all connection values by NaN if 4G is not used
109 Tech = meas_df.NetworkTech; #4=LTE 3=HSPA
110 HSPA_indices = [ i for i in range(len(Tech)) if Tech[i] != '4G' ]
111 RSRP[HSPA_indices] = float('NaN');
112 RSRQ[HSPA_indices] = float('NaN');
113
114 # Replace missing values of RSRQ by NaN, convert to np.float32
115 minus_indices = [ i for i in range(len(RSRQ)) if RSRQ[i] == '-' ]
116 RSRQ[minus_indices] = float('NaN');
117 RSRQ = np.float32(RSRQ)
118 RSRP[minus_indices] = float('NaN');
```

```
RSRP = np.float32(RSRP)

# Coordinate Transforms
[azimuth,elevation,r] = cart2sph(x,y,z)
azimuth = azimuth*180/math.pi
elevation = elevation*180/math.pi

# create data frame with relevant data, remove first entries of RSRP/RSRQ
data_dict = {'x':x,'y':y,'z':z,'azimuth':azimuth,'elevation':elevation,'radius':r,\
'RSRP':RSRP[1:],'RSRQ':RSRQ[1:],'ID number':IDnumberArray[1:]}
df = pandas.DataFrame(data=data_dict)

# OPTIONAL: DUMP ALL DF ENTRIES EXCEPT FOR THOSE OF A CERTAIN TX STATION
id_sel = 0 # set to -1 if no ID should be selected
if id_sel != -1:
    df = df[df['ID number'] == id_sel]
    df = df.drop(columns=['ID number'])  # ID number column no longer needed
    print('\n ID number '+str(id_sel)+' selected. All other entries dumped. \n')


# calculate statistical properties and show them
train_stats = df.describe()
train_stats = train_stats.transpose()
print(train_stats)

# update RSRP and RSRQ after deleting first entry
RSRP = data_dict['RSRP']
RSRQ = data_dict['RSRQ']

print("Reading and preprocessing of data done")
```

Machine learning model

```
"""
Script 2: Machine Learning Model
A neural network with one hidden layer is used to model signal reception properties
"""

import tensorflow as tf
import seaborn as sb
from random import randint

```

```
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.layers import LeakyReLU
from tensorflow.keras.layers import Dense as Dense
from tensorflow.keras.models import Sequential

import matplotlib.pyplot as plt



"""
Input position type definition (cartesian or spherical)
"""

#input_pos_type = 'cartesian' #cartesian coordinates seem to perform better
input_pos_type = 'spherical'


"""
Target type definition (RSRP or RSRQ)
"""

target_type = 'RSRP'
#target_type = 'RSRQ'



"""
Network property definition and data splitting
"""

#number of nodes in the hidden layer
n_nodes_hl1 = 25
n_nodes_hl2 = 10
n_nodes_hl3 = 8
n_nodes_hl4 = 6
n_nodes_hl5 = 0

# alpha for leaky relu
alpha_lrelu = 0.1
```

```
51
52  # split into test and training data
53  #random_state = randint(0,1000)
54  random_state = 127 #127 works very well!
55  train_dataset = df.sample(frac=0.99,random_state=random_state)
56  test_dataset = df.drop(train_dataset.index)
57
58
59
60  """
61  Function definitions
62  """
63
64  # calculate z-value -> normalize input data to [0,1]
65  def norm(x):
66    return (x - train_stats['mean']) / train_stats['std']
67
68
69
70  """
71  Data visualization - pre ML
72  """
73
74  #pairplot - comment to save time if not needed
75  #sb.pairplot(df,corner = True)
76
77
78
79  """
80  Machine Learning - Data preparation
81  """
82
83  #normalize data such that mu = 0 and sigma^2 = 1 and drop NaN-values
84  normed_train_data = norm(train_dataset).dropna()
85  normed_test_data = norm(test_dataset).dropna()
86
87
88  #split off target values
89  if target_type == 'RSRP':
90      train_labels = normed_train_data.pop('RSRP')
91      test_labels = normed_test_data.pop('RSRP')
```

```
92  if target_type == 'RSRQ':
93      train_labels = normed_train_data.pop('RSRQ')
94      test_labels = normed_test_data.pop('RSRQ')
95
96  #remove input values according to input type choice
97  if input_pos_type == 'cartesian':
98      if target_type == 'RSRQ':
99          del normed_train_data['RSRP']
100     if target_type == 'RSRP':
101         del normed_train_data['RSRQ']
102     del normed_train_data['azimuth']
103     del normed_train_data['elevation']
104     del normed_train_data['radius']
105     if target_type == 'RSRQ':
106         del normed_test_data['RSRP']
107     if target_type == 'RSRP':
108         del normed_test_data['RSRQ']
109     del normed_test_data['azimuth']
110     del normed_test_data['elevation']
111     del normed_test_data['radius']
112
113 if input_pos_type == 'spherical':
114     if target_type == 'RSRQ':
115         del normed_train_data['RSRP']
116     if target_type == 'RSRP':
117         del normed_train_data['RSRQ']
118     del normed_train_data['x']
119     del normed_train_data['y']
120     del normed_train_data['z']
121     if target_type == 'RSRQ':
122         del normed_test_data['RSRP']
123     if target_type == 'RSRP':
124         del normed_test_data['RSRQ']
125     del normed_test_data['x']
126     del normed_test_data['y']
127     del normed_test_data['z']
128
129
130
131
132 """
```

```
133 Machine Learning - Execution
134 """
135
136
137 # model for leaky relu activation function
138 model = keras.Sequential()
139 #first layer
140 model.add(Dense(n_nodes_hl1))
141 model.add(LeakyReLU(alpha=alpha_lrelu))
142 #second layer
143 if n_nodes_hl2 != 0:
144     model.add(Dense(n_nodes_hl2))
145     model.add(LeakyReLU(alpha=alpha_lrelu))
146 #third layer
147 if n_nodes_hl3 != 0:
148     model.add(Dense(n_nodes_hl3))
149     model.add(LeakyReLU(alpha=alpha_lrelu))
150 #fourth layer
151 if n_nodes_hl4 != 0:
152     model.add(Dense(n_nodes_hl4))
153     model.add(LeakyReLU(alpha=alpha_lrelu))
154 #fifth layer
155 if n_nodes_hl5 != 0:
156     model.add(Dense(n_nodes_hl5))
157     model.add(LeakyReLU(alpha=alpha_lrelu))
158 #addition layer
159 model.add(Dense(1))
160 model.compile(loss='mse',
161               optimizer='adam',
162               )
163
164
165 # train the model TAKE CARE FOR BATCH SIZE!
166 EPOCHS = 10000
167 history = model.fit( normed_train_data, train_labels, epochs=EPOCHS, \
168                     validation_split = 0.04, verbose=2, shuffle = True, \
169                     batch_size = 1024)
170
171 #summarize the model (inspect it)
172 model.summary()
173
```

```
174 print("random state "+np.str(random_state))
175 print("Training done")
176
177
178
179 # Visualize model history
180 plt.figure()
181 plt.plot(range(EPOCHS),history.history['val_loss'], label='Validation loss')
182 plt.plot(range(EPOCHS),history.history['loss'], label='Training loss')
183 plt.title('Training / validation loss values')
184 plt.ylabel('Loss value')
185 plt.xlabel('Epoch')
186 plt.legend(loc="upper left")
187 plt.show()
```

Plot and save model history

```
1  #save model history
2  plt.figure()
3  plt.plot(range(EPOCHS),history.history['val_loss'], label='Validation loss')
4  plt.plot(range(EPOCHS),history.history['loss'], label='Training loss')
5  plt.title('Training / validation loss values')
6  plt.ylabel('Loss value')
7  plt.xlabel('Epoch')
8  plt.legend(loc="upper left")
9  plt.savefig('./plots_v2/overleaf/'+np.str(target_type)+'_sel_lrelu_'+ \
10 np.str(n_nodes_hl1)+'_'+np.str(n_nodes_hl2)+'_'+np.str(n_nodes_hl3)+'_'+ \
11 np.str(n_nodes_hl4)+'_'+np.str(EPOCHS)+'epochs_'+np.str(input_pos_type)+ \
12 '_RS_'+np.str(random_state)+'.png',dpi = 300)
```

Plot of results

```
1  """
2  Script 3: Plot of results
3  """
4
5  import matplotlib as mpl
6  import matplotlib.pyplot as plt
7  from mpl_toolkits.mplot3d import Axes3D
8
9  import pandas as pd
10
11 # make the plot interactive
```

```
%matplotlib notebook


"""
Choose which values to plot
"""

plot_RSRP = True
plot_RSRQ = False


"""
Calculate Colorbar Limits
"""

## Calculate NN output values to determine colorbar limits

example_batch = normed_test_data.dropna()
example_result = model.predict(example_batch)

#denormalize data
if target_type == 'RSRP':
    print('NN output')
    NNout = example_result*train_stats['std']['RSRP']+train_stats['mean']['RSRP']

if target_type == 'RSRQ':
    print('NN output')
    NNout = example_result*train_stats['std']['RSRQ']+train_stats['mean']['RSRQ']

## Calculate actual coorbar limits

if target_type == 'RSRP':
    vmin_RSRP = min(min(RSRP),min(NNout))
    vmax_RSRP = max(max(RSRP),max(NNout))
    vmin_RSRQ = min(RSRQ)
    vmax_RSRQ = max(RSRQ)

if target_type == 'RSRQ':
    vmin_RSRQ = min(min(RSRQ),min(NNout))
    vmax_RSRQ = max(max(RSRQ),max(NNout))
    vmin_RSRP = min(RSRP)
```

```
53      vmax_RSRP = max(RSRP)
54
55
56  """
57  Plot actual flights
58  """
59
60  # PLOT 1: MEASURED DATA - RSRP
61  if plot_RSRP == True:
62      #plot initialization
63      fig = plt.figure(dpi=150)
64      fig.suptitle('RSRP - Measured Data', fontsize=16)
65      ax = fig.add_subplot(111, projection='3d')
66      ax.set_xlabel('X')
67      ax.set_ylabel('Y')
68      ax.set_zlabel('Z')
69      #draw dots, colored according to RSRP
70      scat = ax.scatter(x,y,z,c=RSRP,s=5,vmin=vmin_RSRP,vmax=vmax_RSRP)
71      #transmitter position
72      ax.scatter(0,0,0,c='blue',s=100)
73      ax.text(0, 0, 0, "Transmitter/zero", color='blue')
74      #start position
75      ax.scatter(x[0],y[0],z[0],c='red',s=100)
76      ax.text(x[0],y[0],z[0], "Start", color='red')
77      #colorbar
78      cbar = fig.colorbar(scat,label = "RSRP")
79      #show plot
80      plt.show()
81
82
83  # PLOT 2: MEASURED DATA - RSRQ
84  if plot_RSRQ == True:
85      #plot initialization
86      fig = plt.figure(dpi=150)
87      fig.suptitle('RSRQ - Measured Data', fontsize=16)
88      ax = fig.add_subplot(111, projection='3d')
89      ax.set_xlabel('X')
90      ax.set_ylabel('Y')
91      ax.set_zlabel('Z')
92      #draw dots, colored according to RSRQ
93      scat = ax.scatter(x,y,z,c=RSRQ,s=5,vmin=vmin_RSRQ,vmax=vmax_RSRQ)
```

```
94      #transmitter position
95      ax.scatter(0,0,0,c='blue',s=100)
96      ax.text(0, 0, 0, "Transmitter/zero", color='blue')
97      #start position
98      ax.scatter(x[0],y[0],z[0],c='red',s=100)
99      ax.text(x[0],y[0],z[0], "Start", color='red')
100     #colorbar
101     fig.colorbar(scat,label = "RSRQ")
102     #show plot
103     plt.show()
104
105
106
107 """
108 Plot neural network outputs
109 In the plots, NN output data that stems from test input data (i.e. data that the \
110 network has not "seen" yet) is plotted larger
111 """
112
113 #Calculation NN outputs
114 NNout_test_normed = model.predict(normed_test_data.dropna())
115 NNout_train_normed = model.predict(normed_train_data.dropna())
116
117 #denormalize data
118 NNout_test = NNout_test_normed*train_stats['std'][target_type]+ \
119                         train_stats['mean'][target_type]
120 NNout_train = NNout_train_normed*train_stats['std'][target_type]+ \
121                        train_stats['mean'][target_type]
122
123 #coordinates of test data
124 x_test = test_dataset.dropna()['x']
125 y_test = test_dataset.dropna()['y']
126 z_test = test_dataset.dropna()['z']
127
128 #coordinates of train data
129 x_train = train_dataset.dropna()['x']
130 y_train = train_dataset.dropna()['y']
131 z_train = train_dataset.dropna()['z']
132
133 #plot data
134 #plot initialization
```

```
fig = plt.figure(dpi=150)
fig.suptitle(str(target_type)+' - Estimated Value by Neural Network', fontsize=16)
ax = fig.add_subplot(111, projection='3d')
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')
#draw dots, colored according to RSRP/RSRQ, test data is drawn bigger
if target_type == 'RSRP':
    scat_test = ax.scatter(x_test,y_test,z_test,c=NNout_test.T[0],s=30,vmin=vmin_RSRP \
                           ,vmax=vmax_RSRP)
    scat_train = ax.scatter(x_train,y_train,z_train,c=NNout_train.T[0],s=5 \
                           ,vmin=vmin_RSRP,vmax=vmax_RSRP)
if target_type == 'RSRQ':
    scat_test = ax.scatter(x_test,y_test,z_test,c=NNout_test.T[0],s=30,vmin=vmin_RSRQ, \
                           vmax=vmax_RSRQ)
    scat_train = ax.scatter(x_train,y_train,z_train,c=NNout_train.T[0],s=5, \
                           vmin=vmin_RSRQ,vmax=vmax_RSRQ)
#transmitter position
ax.scatter(0,0,0,c='blue',s=100)
ax.text(0, 0, 0, "Transmitter/zero", color='blue')
#start position
ax.scatter(x[0],y[0],z[0],c='red',s=100)
ax.text(x[0],y[0],z[0], "Start", color='red')
#colorbar
fig.colorbar(scat_test,label = str(target_type))
#show plot
plt.show()
```

Save model

```
#save model as json
#codes from
# https://machinelearningmastery.com/save-load-keras-deep-learning-models/

# serialize model to JSON
model_json = model.to_json()
with open("./models_v2/model_RSRP_sel_lrelu_25_10_8_6_10kepochs_BS1024.json", "w") \
    as json_file:
    json_file.write(model_json)
# serialize weights to HDF5
model.save_weights("./models_v2/model_RSRP_sel_lrelu_25_10_8_6_10kepochs_BS1024.h5")
print("Saved model to disk")
```

Load model

```
# load saved model
from tensorflow import keras
from tensorflow.keras.models import model_from_json

# load json and create model
json_file = open('./models_v2/model_RSRQ_sel_lrelu_25_10_8_6_10kepochs.json', 'r')
loaded_model_json = json_file.read()
json_file.close()
loaded_model = model_from_json(loaded_model_json)
# load weights into new model
loaded_model.load_weights("./models_v2/model_RSRQ_sel_lrelu_25_10_8_6_10kepochs.h5")
print("Loaded model from disk")

# choose correct values MANUALLY!
model = loaded_model
input_pos_type = 'cartesian'
target_type = 'RSRQ'
```

```
"""
EXECUTE THIS SCRIPT AFTER LOADING A MODEL TO ENABLE PLOTTING OF FLIGHTS
"""

import tensorflow as tf
import seaborn as sb

from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.layers import LeakyReLU
from tensorflow.keras.layers import Dense as Dense
from tensorflow.keras.models import Sequential

import matplotlib.pyplot as plt




"""
Input position type definition (cartesian or spherical)
"""

input_pos_type = 'cartesian' #cartesian oordinates seem to perform better
```

```
#input_pos_type = 'spherical'


"""
Target type definition (RSRP or RSRQ)
"""

#target_type = 'RSRP'
target_type = 'RSRQ'



"""
Network property definition and data splitting
"""

#number of nodes in the hidden layer
n_nodes_hl1 = 25
n_nodes_hl2 = 10
n_nodes_hl3 = 8
n_nodes_hl4 = 6
n_nodes_hl5 = 0

# alpha for leaky relu
alpha_lrelu = 0.1

# split into test and training data
train_dataset = df.sample(frac=0.99,random_state=0)
test_dataset = df.drop(train_dataset.index)



"""
Function definitions
"""

# calculate z-value -> normalize input data to [0,1]
def norm(x):
  return (x - train_stats['mean']) / train_stats['std']


```

```
65
66  """
67  Data visualization - pre ML
68  """
69
70  #pairplot - comment to save time if not needed
71  #sb.pairplot(df,corner = True)
72
73
74
75  """
76  Machine Learning - Data preparation
77  """
78
79  #normalize data such that mu = 0 and sigma^2 = 1 and drop NaN-values
80  normed_train_data = norm(train_dataset).dropna()
81  normed_test_data = norm(test_dataset).dropna()
82
83
84  #split off target values
85  if target_type == 'RSRP':
86      train_labels = normed_train_data.pop('RSRP')
87      test_labels = normed_test_data.pop('RSRP')
88  if target_type == 'RSRQ':
89      train_labels = normed_train_data.pop('RSRQ')
90      test_labels = normed_test_data.pop('RSRQ')
91
92  #remove input values according to input type choice
93  if input_pos_type == 'cartesian':
94      if target_type == 'RSRQ':
95          del normed_train_data['RSRP']
96      if target_type == 'RSRP':
97          del normed_train_data['RSRQ']
98      del normed_train_data['azimuth']
99      del normed_train_data['elevation']
100     del normed_train_data['radius']
101     if target_type == 'RSRQ':
102         del normed_test_data['RSRP']
103     if target_type == 'RSRP':
104         del normed_test_data['RSRQ']
105     del normed_test_data['azimuth']
```

```
106         del normed_test_data['elevation']
107         del normed_test_data['radius']
108 
109     if input_pos_type == 'spherical':
110         if target_type == 'RSRQ':
111             del normed_train_data['RSRP']
112         if target_type == 'RSRP':
113             del normed_train_data['RSRQ']
114         del normed_train_data['x']
115         del normed_train_data['y']
116         del normed_train_data['z']
117         if target_type == 'RSRQ':
118             del normed_test_data['RSRP']
119         if target_type == 'RSRP':
120             del normed_test_data['RSRQ']
121         del normed_test_data['x']
122         del normed_test_data['y']
123         del normed_test_data['z']
124 
125 
126     # update RSRP and RSRQ after deleting first entry
127     RSRP = df['RSRP']
128     RSRQ = df['RSRQ']
129     x = df['x']
130     y = df['y']
131     z = df['z']
```

Plausibility check

```
1   """
2   Plot plausibility check
3   """
4 
5   import matplotlib as mpl
6   import matplotlib.pyplot as plt
7   from mpl_toolkits.mplot3d import Axes3D
8 
9 
10  import pandas as pd
11 
12  def cart2sph(x, y, z):
13      hxy = np.hypot(x, y)
```

```
    r = np.hypot(hxy, z)
    el = np.arctan2(z, hxy)
    az = np.arctan2(y, x)
    return az, el, r

def norm(x):
  return (x - plaus_coords_df_stats['mean']) / plaus_coords_df_stats['std']


plaus_coords = np.zeros((1,3))
plot_coords = np.zeros((1,3))

#calculate NN outputs for test grid
#plaus_coords = np.array([0,0,0])
for x in range(-350,350,50):
    for y in range(-350,350,50):
        for z in range(0,200,50):
            if input_pos_type == 'spherical':
                [azimuth,elevation,r] = cart2sph(x,y,z)
                azimuth = azimuth*180/math.pi
                elevation = elevation*180/math.pi
                arr = np.array([azimuth,elevation,r])
            if input_pos_type == 'cartesian':
                arr = np.array([x,y,z])
            plaus_coords = np.append(plaus_coords, [arr], axis = 0)
            xyz = np.array([x,y,z])
            plot_coords = np.append(plot_coords,[xyz], axis = 0)

plaus_coords_df = pd.DataFrame(data=plaus_coords, columns=["x/az", "y/el","z/r"])
plaus_coords_df_stats = plaus_coords_df.describe()
plaus_coords_df_stats = plaus_coords_df_stats.transpose()

plaus_coords_normed = norm(plaus_coords_df).dropna()


#Calculation NN outputs
NNout_plaus_normed = model.predict(plaus_coords_normed)

#denormalize data
NNout_plaus = NNout_plaus_normed*train_stats['std'][target_type]+ \
                train_stats['mean'][target_type]
```

```
55
56  ## Calculate colorbar limits
57  if target_type == 'RSRP':
58      vmin_RSRP = min(min(RSRP),min(NNout_plaus))
59      vmax_RSRP = max(max(RSRP),max(NNout_plaus))
60      vmin_RSRQ = min(RSRQ)
61      vmax_RSRQ = max(RSRQ)
62  if target_type == 'RSRQ':
63      vmin_RSRQ = min(min(RSRQ),min(NNout_plaus))
64      vmax_RSRQ = max(max(RSRQ),max(NNout_plaus))
65      vmin_RSRP = min(RSRP)
66      vmax_RSRP = max(RSRP)
67
68  #get coordinates from array
69  x_grid = plot_coords[:,0]
70  y_grid = plot_coords[:,1]
71  z_grid = plot_coords[:,2]
72
73  #plot initialization
74  fig = plt.figure(dpi = 200)
75  fig.suptitle(str(target_type)+' - Plausibility check', fontsize=16)
76  ax = fig.add_subplot(111, projection='3d')
77  ax.set_xlabel('X')
78  ax.set_ylabel('Y')
79  ax.set_zlabel('Z')
80  #draw dots, colored according to RSRP/RSRQ, test data is drawn bigger
81  if target_type == 'RSRP':
82      scat_plaus = ax.scatter(x_grid,y_grid,z_grid,c=NNout_plaus.T[0],s=10,\
83      vmin=vmin_RSRP,vmax=vmax_RSRP)
84  if target_type == 'RSRQ':
85      scat_plaus = ax.scatter(x_grid,y_grid,z_grid,c=NNout_plaus.T[0],s=10,\
86      vmin=vmin_RSRQ,vmax=vmax_RSRQ)
87  #transmitter position
88  ax.scatter(0,0,0,c='blue',s=100)
89  ax.text(0, 0, 0, "Zero", color='blue')
90  #colorbar
91  fig.colorbar(scat_plaus,label = str(target_type))
92  #show plot
93  plt.show()
```

Plot high resolution learning curve (for documentation purposes)

```
plt.figure(dpi=300)
plt.plot(range(EPOCHS),history.history['val_loss'], label='Validation loss')
plt.plot(range(EPOCHS),history.history['loss'], label='Training loss')
plt.title('Training / validation loss values')
plt.ylabel('Loss value')
plt.xlabel('Epoch')
plt.legend(loc="upper left")
plt.show()
```

MA_v3_optimizer.ipynb

Optimizer - data preprocessing

```
"""
OPTIMIZER - USE THIS SCRIPT TO ITERATE THROUGH INITIALIZATION SEEEDS

Script 1: Reading and preprocessing of data from txt files

TO LOOK UP CELLS: go to http://www.opencellid.org
                  search for pirching an der raab
"""

import pandas
import numpy as np
import math




"""
Function definitions
"""

def cart2sph(x, y, z):
    hxy = np.hypot(x, y)
    r = np.hypot(hxy, z)
    el = np.arctan2(z, hxy)
    az = np.arctan2(y, x)
    return az, el, r


```

```
"""
Get measurement data from csv file
"""

#Measurement dataframe
meas_df = pandas.read_csv('Messdaten3.csv', sep=';')

#Connection values, use float to enable NaN
#throw away first values later!
RSRP = pandas.Series(meas_df.Level,dtype='float32')
RSRQ = pandas.Series(meas_df.Qual,dtype='float32')



"""
Process measurement data and prepare it for further usage
"""

#Position Values
LongitudeMobile = meas_df.Longitude
LatitudeMobile = meas_df.Latitude
HeightMobile = meas_df.Height

#get rid of excess dots
LongitudeMobileArray = np.zeros(len(LongitudeMobile)-1)
for i in range(len(LongitudeMobile)-1):
    test = LongitudeMobile[i+1]
    newstr = test[:6] + test[7:]
    LongitudeMobileArray[i] = float(newstr)
LatitudeMobileArray = np.zeros(len(LatitudeMobile)-1)
for i in range(len(LatitudeMobile)-1):
    test = LatitudeMobile[i+1]
    newstr = test[:6] + test[7:]
    LatitudeMobileArray[i] = float(newstr)

#preprocess height
HeightMobileArray = np.zeros(len(HeightMobile)-1)
minheight = min(HeightMobile[1:]) # throw away first three values
for i in range(1,len(HeightMobile)):
    h = HeightMobile[i]
    HeightMobileArray[i-1] = float(h-minheight)
```

```
z = HeightMobileArray

#Conversion from coordinates to meters - mean is chosen as zero
x = (LongitudeMobileArray - np.mean(LongitudeMobileArray))*76000;
y = (LatitudeMobileArray - np.mean(LatitudeMobileArray))*111000;
#Factors obtained from website:
#https://rechneronline.de/geo-koordinaten/#entfernung


# create dict of raw cell IDs
cellIDs = meas_df.RAWCELLID
IDnumber = 0 #number that is assigned to each unique cell ID, starting with zero
IDdict = {} #dictionary containing cell IDs and their respective numbers
for i in range(len(cellIDs)):
    if not cellIDs[i] in IDdict:
        IDdict[cellIDs[i]] = IDnumber
        IDnumber = IDnumber + 1
IDnumberMax = IDnumber
print("ID dictionary:")
print(IDdict)

# create array of ID numbers
#IDnumberArray = np.zeros(len(cellIDs))   # np array
IDnumberArray = []                        # python array
for i in range(len(cellIDs)):
    #IDnumberArray[i] = int(IDdict[cellIDs[i]])     # np array
    IDnumberArray.append(int(IDdict[cellIDs[i]]))   # python array

# check for number of station appearances
stationCount = 0
for j in range(IDnumberMax):
    for i in range(len(cellIDs)):
        if IDnumberArray[i] == j:
            stationCount = stationCount + 1
    print("IDnumber "+str(j)+" appears "+str(stationCount)+" times.")
    stationCount = 0


# Replace all connection values by NaN if 4G is not used
Tech = meas_df.NetworkTech; #4=LTE 3=HSPA
HSPA_indices = [ i for i in range(len(Tech)) if Tech[i] != '4G' ]
```

```
RSRP[HSPA_indices] = float('NaN');
RSRQ[HSPA_indices] = float('NaN');

# Replace missing values of RSRQ by NaN, convert to np.float32
minus_indices = [ i for i in range(len(RSRQ)) if RSRQ[i] == '-' ]
RSRQ[minus_indices] = float('NaN');
RSRQ = np.float32(RSRQ)
RSRP[minus_indices] = float('NaN');
RSRP = np.float32(RSRP)

# Coordinate Transforms
[azimuth,elevation,r] = cart2sph(x,y,z)
azimuth = azimuth*180/math.pi
elevation = elevation*180/math.pi

# create data frame with relevant data, remove first entries of RSRP/RSRQ
data_dict = {'x':x,'y':y,'z':z,'azimuth':azimuth,'elevation':elevation,'radius':r,\
             'RSRP':RSRP[1:],'RSRQ':RSRQ[1:],'ID number':IDnumberArray[1:]}
df = pandas.DataFrame(data=data_dict)

# OPTIONAL: DUMP ALL DF ENTRIES EXCEPT FOR THOSE OF A CERTAIN TX STATION
id_sel = 0 # set to -1 if no ID should be selected
if id_sel != -1:
    df = df[df['ID number'] == id_sel]
    df = df.drop(columns=['ID number'])  # ID number column no longer needed
    print('\n ID number '+str(id_sel)+' selected. All other entries dumped. \n')


# calculate statistical properties and show them
train_stats = df.describe()
train_stats = train_stats.transpose()
print(train_stats)

# update RSRP and RSRQ after deleting first entry
RSRP = data_dict['RSRP']
RSRQ = data_dict['RSRQ']

print("Reading and preprocessing of data done")
```

Optimizer - actual optimization

```
"""
Script 2: Machine Learning Model
A neural network with one hidden layer is used to model signal reception properties
"""

import tensorflow as tf
import seaborn as sb
from random import randint

from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.layers import LeakyReLU
from tensorflow.keras.layers import Dense as Dense
from tensorflow.keras.models import Sequential

import matplotlib.pyplot as plt




"""
Input position type definition (cartesian or spherical)
"""

#input_pos_type = 'cartesian' #cartesian oordinates seem to perform better
input_pos_type = 'spherical'


"""
Target type definition (RSRP or RSRQ)
"""

target_type = 'RSRP'
#target_type = 'RSRQ'



"""
Network property definition and data splitting
"""

```

```
42  #number of nodes in the hidden layer
43  n_nodes_hl1 = 25
44  n_nodes_hl2 = 10
45  n_nodes_hl3 = 8
46  n_nodes_hl4 = 6
47  n_nodes_hl5 = 0
48
49  # alpha for leaky relu
50  alpha_lrelu = 0.1
51
52  for ii in range(100):
53      # split into test and training data
54      random_state = ii
55      train_dataset = df.sample(frac=0.99,random_state=random_state)
56      test_dataset = df.drop(train_dataset.index)
57
58
59
60      """
61      Function definitions
62      """
63
64      # calculate z-value -> normalize input data to [0,1]
65      def norm(x):
66        return (x - train_stats['mean']) / train_stats['std']
67
68
69
70      """
71      Data visualization - pre ML
72      """
73
74      #pairplot - comment to save time if not needed
75      #sb.pairplot(df,corner = True)
76
77
78
79      """
80      Machine Learning - Data preparation
81      """
82
```

```
    #normalize data such that mu = 0 and sigma^2 = 1 and drop NaN-values
    normed_train_data = norm(train_dataset).dropna()
    normed_test_data = norm(test_dataset).dropna()


    #split off target values
    if target_type == 'RSRP':
        train_labels = normed_train_data.pop('RSRP')
        test_labels = normed_test_data.pop('RSRP')
    if target_type == 'RSRQ':
        train_labels = normed_train_data.pop('RSRQ')
        test_labels = normed_test_data.pop('RSRQ')

    #remove input values according to input type choice
    if input_pos_type == 'cartesian':
        if target_type == 'RSRQ':
            del normed_train_data['RSRP']
        if target_type == 'RSRP':
            del normed_train_data['RSRQ']
        del normed_train_data['azimuth']
        del normed_train_data['elevation']
        del normed_train_data['radius']
        if target_type == 'RSRQ':
            del normed_test_data['RSRP']
        if target_type == 'RSRP':
            del normed_test_data['RSRQ']
        del normed_test_data['azimuth']
        del normed_test_data['elevation']
        del normed_test_data['radius']

    if input_pos_type == 'spherical':
        if target_type == 'RSRQ':
            del normed_train_data['RSRP']
        if target_type == 'RSRP':
            del normed_train_data['RSRQ']
        del normed_train_data['x']
        del normed_train_data['y']
        del normed_train_data['z']
        if target_type == 'RSRQ':
            del normed_test_data['RSRP']
        if target_type == 'RSRP':
```

```
            del normed_test_data['RSRQ']
        del normed_test_data['x']
        del normed_test_data['y']
        del normed_test_data['z']



    """
    Machine Learning - Execution
    """


    # model for leaky relu activation function
    model = keras.Sequential()
    #first layer
    model.add(Dense(n_nodes_hl1))
    model.add(LeakyReLU(alpha=0.1))
    #second layer
    if n_nodes_hl2 != 0:
        model.add(Dense(n_nodes_hl2))
        model.add(LeakyReLU(alpha=alpha_lrelu))
    #third layer
    if n_nodes_hl3 != 0:
        model.add(Dense(n_nodes_hl3))
        model.add(LeakyReLU(alpha=alpha_lrelu))
    #fourth layer
    if n_nodes_hl4 != 0:
        model.add(Dense(n_nodes_hl4))
        model.add(LeakyReLU(alpha=alpha_lrelu))
    #fifth layer
    if n_nodes_hl5 != 0:
        model.add(Dense(n_nodes_hl5))
        model.add(LeakyReLU(alpha=alpha_lrelu))
    #addition layer
    model.add(Dense(1))
    model.compile(loss='mse',
                  optimizer='adam',
                  )

    print("training started with random state "+np.str(random_state))
```

```
165
166     # train the model TAKE CARE FOR BATCH SIZE!
167     EPOCHS = 2000
168     history = model.fit( normed_train_data, train_labels, epochs=EPOCHS, \
169                         validation_split = 0.04, verbose=1, shuffle = True, batch_size = 1)
170
171     #summarize the model (inspect it)
172     model.summary()
173
174     plt.figure()
175     plt.plot(range(EPOCHS),history.history['val_loss'], label='Validation loss')
176     plt.plot(range(EPOCHS),history.history['loss'], label='Training loss')
177     plt.title('Training / validation loss values')
178     plt.ylabel('Loss value')
179     plt.xlabel('Epoch')
180     plt.legend(loc="upper left")
181     plt.savefig('./plots_v2/optimizer/REPROD2'+np.str(target_type)+'_sel_lrelu_'+\
182             np.str(n_nodes_hl1)+'_'+np.str(n_nodes_hl2)+'_'+np.str(n_nodes_hl3)+\
183             '_'+np.str(n_nodes_hl4)+'_'+np.str(EPOCHS)+'epochs_'+np.str(input_pos_type)+\
184             '_RS_'+np.str(random_state)+'.png',dpi = 300)
185     plt.show()
186
187
188
189
190
191
192 print("Training done")
```