Final report AE4350: Bio-inspired AI for AE applications

ESA Lunar Lava Tubes detection concept with gravity surveys

Name Frank de Veld

Date of Delivery September 1st, 2020
Image courtesy of ESA

Delft University of Technology

CONTENTS

List of Figures

List of Tables

1

LIST OF SYMBOLS AND ABBREVIATIONS

Symbol	Meaning
R	Radius
$\Delta\rho$ or ρ	Density (difference)
z	Cavity depth
x	Distance to maximum anomaly
G	Universal gravitational constant
g	Gravity anomaly
g_n	Normalised gravity anomaly
S_m	Size multiplication
Δg	Bouguer anomaly
γ	Normal gravity
δg_h	Free-air reduction
δg_{bpr}	Bouguer plate reduction
δg_{ter}	Terrain reduction
$_i$	Input layer
$_j$	Hidden layer
$_k$	Output layer
E	Total error
y	Final output
X	Initial input
v	Layer input
w	Weight
b	Bias
ϕ	Activation function
ϵ	Learning rate
V	Vertical
H	Horizontal

Table 1.1: Symbols and their meaning used throughout the report

Abbreviation	Meaning
ESA	European Space Agency
ESTEC	European Space Research and Technology Centre (at Noordwijk, the Netherlands)
NASA	National Aeronautics and Space Administration
JAXA	Japan Aerospace Exploration Agency
CSA	Canadian Space Agency
MHP/MHS	Marius Hill Pit/Marius Hill Skylight
TRL	Technology Readiness Level
SELENE	Selenological and Engineering Explorer; JAXA lunar satellite
LRO	Lunar Reconnaissance Orbiter; NASA lunar satellite
GRAIL	Gravity Recovery and Interior Laboratory; NASA lunar satellite
GPR	Ground-Penetrating Radar
ANN	Artificial Neural Network
NN	Neural Network
IGMAS	Interactive Geophysical Modelling Assistant

Table 1.2: Abbreviations and their meaning used throughout the report

2

PREFACE

This report is the result of an ongoing six-month internship at Queen's University, Kingston, Canada, by the author Frank de Veld. Parts of this internship on lunar lava tube detection with gravity surveying have been heavily relying on the use of artificial intelligence, specifically (self)-supervised neural networks. Use has been made from the knowledge taught in the course AE4350 Bio-inspired Intelligence and learning for Aerospace Applications at TU Delft. Due to the clear relevancy of this project to this course, the following report is handed in as part of the final project of this course. The report is an adapted version of documentation written during the mentioned internship, focusing on the AI-side of the project and having a structural approach which is expected from a final assignment for such a course.

3

INTRODUCTION

The mentioned internship has been funded by SysNova, an initiative by the European Space Agency in order to structurally asses technologies and space mission concepts. These ideas usually come from industry or academia and are selected in some kind of contest scheme. Such a contest scheme has also been started previous year, with the following problem statement:

"This Campaign addresses the technical, operational and instrumentation challenges driving new mission concepts to enable the detection, mapping and robotic exploration of lunar caves."

This statement has lead to Canadian rover company Canadensys making a joint proposal with the geoscience department of Queen's University, Kingston to investigate the potential a surface-based rover with a gravitometer in such a lunar exploration situation. This would be backed by theoretical research with simulations to show what sorts of lunar situations could be expected and how they would be observed. This proposal has been accepted by ESA in the beginning of 2020, together with four other teams, which are expected to make a collaborative joint proposal at the end of the six-month project [5].

With the project starting, the goals for my internship were defined, roughly coming down to the following two parts:

- Modelling cavities in gravity software to investigate resulting gravity signals and measurements, hereby looking at the limits of detection, a process known as *forward modelling*
- Without considering the modelled situation, using the simulated gravity measurements to trace back what the subsurface cavity looks like, a process known as *inverse modelling*

While forward modelling is relatively straightforward and an exact procedure, inverse modelling is far from that. Lack of knowledge of the situation, noisy measurements and the ambiguity of several configurations leads to inverse modelling being very similar to statistics and optimisation. For both forward and inverse modelling, dozens of methods exist. For forward modelling, these are mostly regarding efficiency of obtaining results, while for inverse modelling these different methods exist to improve accuracy in distinct situations. Yet, efficiency and speed are also still very important, especially in the (semi-)autonomous situation for lunar rovers, which is what ESA is striving for in this project. In this report, the use of artificial intelligence, specifically artificial neural networks, in gravity data interpretation is investigated. Many authors have done similar research for terrestrial applications, for example by Grêt *et al.* [6], Salem *et al.* [7], Hajian *et al.* [8] and Eshaghzadeh and Hajian [9]. These application can be seen as classification algorithms, as the cavities to be identified are always elementary shapes, and only a few parameters are to be investigated. However, this is exactly the power of artificial neural networks in this case, with them being able to see basic structures and patterns.

Especially in this application of preliminary cavity probing, such a classification is perfect. It is not needed to get a detailed map of the cavity, but rather a first idea where cavities could be located. A follow-up rover then would enter the cavity through a so called *sky-light*, a steep vertical hole thought to be a result of partial collapse, and afterwards this rover will explore the caves further, for example with the laser-scanning method of LIDAR. Thus, the goal of the surface gravity survey boils down to two things; firstly whether the cave is 'worth exploring' and secondly whether the cave is accessible. The first remark is regarding the difference between a spherical cavity, created by for example ancient lava outpouring, and a horizontally extended cylindrical cavity, created by lava flow. Only the second, horizontal, cavity is really considered of interest. The second remark regarding accessibility is mostly about the depth of the cavity; from a certain depth on, it is not possible for the lunar rovers to safely explore these caves. Thus, the gravity interpretation algorithm should mostly give answers to these two questions, secondary information is of lesser importance.

There are several reasons why the use of neural networks here is innovative and worth investigating. First of all, they use no assumptions on the underlying situation and have no connection with existing mathematical algorithms for inversion modelling. Thus, the results of the neural network serve as an excellent extra check for the existing inversion algorithms. They are in no way meant as a replacement, as there are also important disadvantages, but as an add-on. This is possible as the required computational power to *use* a neural network is very small, certainly compared to the rather complicated inversion process of most inversion algorithm. Of course, the process of *training*, *developing* and *testing* the neural network costs a lot of time and consumes much computational power. Yet, when the neural network is put into use, it requires very little. This is especially relevant for (semi-)autonomous applications with mass constraints, such as this ESA concept. What also makes this neural network application particularly suitable is that the output is exactly what is desired

in this situation, namely the depth and the shape of the cavity; this is explained later in Chapter 4. Of course there are also important limitations; a crucial one is that neural networks behave as 'black boxes', meaning the inner behaviour is largely unknown, and in the case of lacking performance, improving this is not straightforward. While generally robust and reliable, the accuracy can be lacking. These observations mean that the artificial neural network is not fit for the role of main interpreter of gravity data results, though it can serve as a back-up for potential verification.

With the topic introduced, more detailed information is provided in Chapter 4 as well as required theory. After this, Chapter 5 talks about the steps taken in the working process and the development of the neural network and project as a whole. Here, intermediary results and a quantitative analysis are also given. Chapter 6 summarises the most important results, after which the conclusion follows in Chapter 7 and discussion and further recommendations in Chapter 8. Lastly, there is a reference section as well as appendices for short code explanations. All code can be found here.

4

BACKGROUND THEORY

In order to fully understand the context of the project, one must learn about relevant gravity terminology as well as theory to fully see the power of the use of an artificial neural network in this context. This chapter concerns background theory of microgravity, the area of gravity surveying concerning these typical small differences, as well as the mission concept background and some theory on artificial neural networks.

4.1. THE ORIGIN OF LAVA TUBES

As this project is about lunar lava tube classification, insight is needed in the characteristics of these lava tubes. These are the target of the gravity surveys, as well as the objects which are modelled via forward modelling. While the current Moon is fairly inactive in terms of geophysical processes, its origin is thought to be extremely active, with many volcanoes present ejecting material and gasses to the surface, likely even creating a relatively thick atmosphere. Magma oceans were also present during the origin of the Moon, but also after this magma had solidified, much volcanic activity was present. Current-day signs of that are visible in lunar sinuous rilles, valleys in the lunar crust, and in the dark *mare*, the result of large lava upwellings and coverings. Another feature that is likely to exist, but has not been found yet, is a lava tube. Just like on Earth, these structures are voids left behind by underground lava flows which have since disappeared. The tubes have been covered by layers of other magma outflows or meteorites. If they were perfectly preserved, they would be difficult to detect and even more difficult to enter, but many features known as *skylights* have been found. These pits in the surface are remarkably deep and don't resemble craters from meteorites, as there is no rubble around the pits and the ratio between radius and depth is much off. It is thought that these skylights are collapsed parts of lava tubes, providing an entrance to them.

There are other reasons why researchers expect the presence of lunar lava tubes at these locations. First of all, these skylights are often grouped together in curved or straight lines, sometimes accompanied by sinuous rilles, as shown in Figure 4.1, obtained from Hurwitz *et al.* [1]. Other observations are regarding orbital gravity data from the GRAIL spacecrafts, which measured the gravity field of the Moon to an unprecedented accuracy, much higher than the gravity field measurements of Earth. Near the Marius Hill Skylight, lateral mass deficits were observed, which could indicate long-stretching lunar lava tubes [10]. Observations from orbit can aid in finding subsurface lava tubes, but only when the feature is very significant and multiple kilometers in length, and the depth information is difficult if not impossible to extract from these measurements [11] [12] [13].

The Marius Hill Skylight has also been investigated with other instruments, for example the lunar radar sounder on board of the SELENE spacecraft. Interpretation by Kaku *et al.* [10] of this data yielded many potential subsurface lava tubes, yet since the dielectric constant of the lunar regolith is not very well known and could very a lot depending on position, this does not serve as proof.

Interestingly enough, a ground-penetrating radar has also been applied to the Marius Hills region, but based from Earth. The research by Campbell *et al.* [14] included an observatory in Puerto Rico aiming a circular polarised signal to the Marius Hills region, and the partially reflected, partially absorbed signal was received by an observatory in West Virginia. The resolution was 500 meter, barely enough to detect a subsurface cave, but as their research was focused on basalt material, subsurface cavities were not clearly found. However, it shows that with such a technique and perhaps a better resolution, low-cost investigations from Earth could be performed (only if the region of interest is on the Earth-facing side of the Moon).

There are a number of general features of lunar lava tubes, which can differ from terrestrial lava tubes [15] [16] [17]:

- Lunar lava tubes are expected to be much larger than their terrestrial counterparts; where lava tubes on Earth are rarely larger than a few tens of meters in radius, lunar lava tubes are expected to be at least this size, possibly up to a few kilometers in radius. This is partially due to the lower gravitational acceleration and partially due to a different origin
- The floors and walls are expected to be very rough; glassy, smooth walls like on Earth are not expected
- Stability is two-fold: on one hand, the structures have been existent over billions of years, on the other hand, the skylights themselves are signs of instability. Seismic activity, meteor impacts and drastic temperature changes combined with possible robotic or human activity can be fatal for the lava tube.

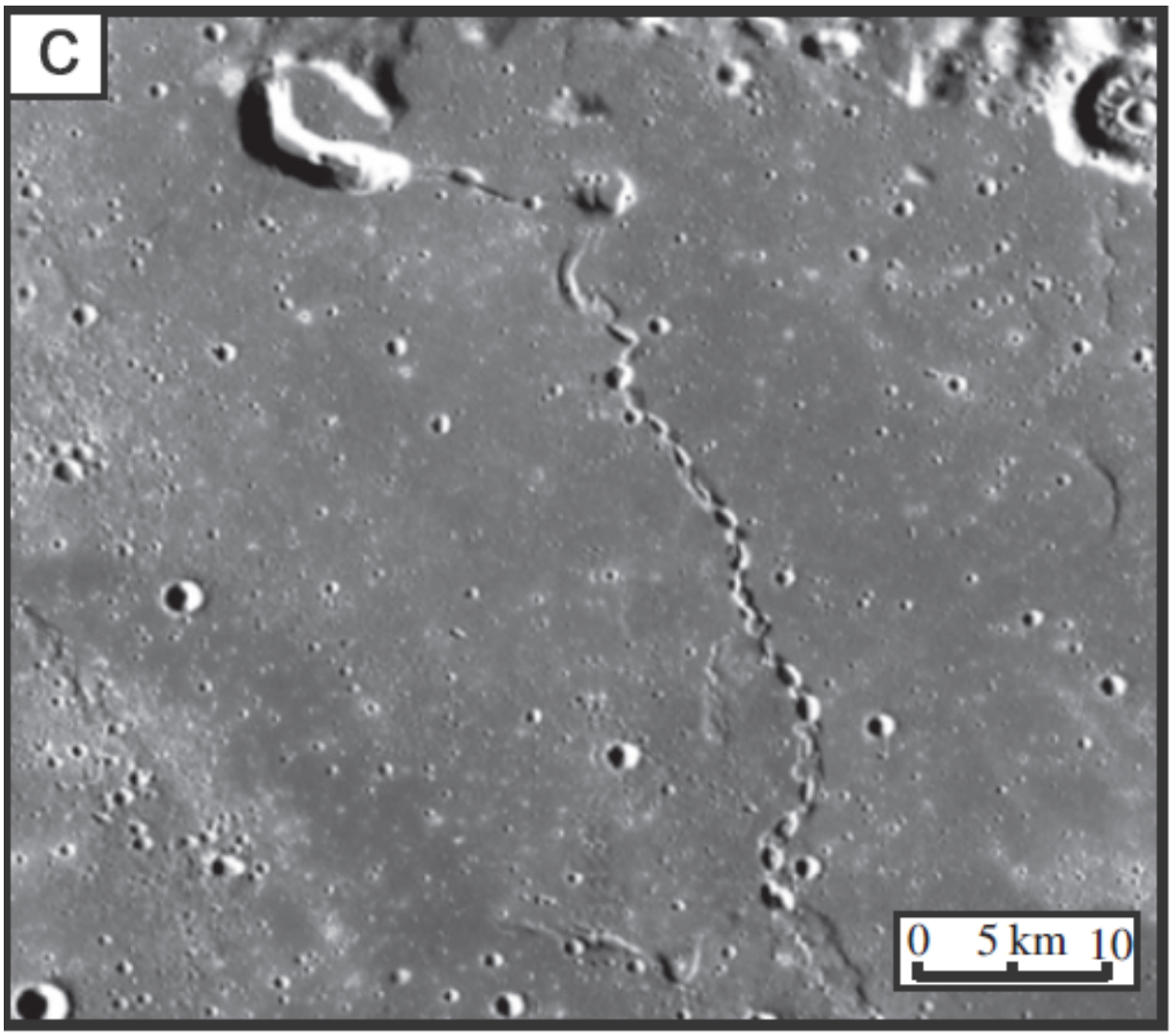


Figure 4.1: The peculiar grouping of skylights and a local depression, possibly indicating a subsurface lunar lava tube [1]

4.2. MISSION CONTEXT

Though ESA's SysNova has specified few details about the mission concept, they are crucial to abide to. Their vision shows a mission encompassing one or several rovers in a mission duration up to fourteen days; one full lunar day. These rovers should use their sensors and devices to explore a subsurface cavity, specifically a lunar lava tube, by entering through a collapsed area known as a skylight. For this investigation, the Marius Hill Pit in the Marius Hills on the Moon is used as the proposed site of investigation. It is a place about which thorough research has been done, and the idea is that indeed a lava tube is buried beneath the surface [18].

The joint proposal of Canadensys and the department of geological sciences and geological engineering of Queen's University is to develop a "rover-based system for scouting and mapping lava tubes from the Moon's surface using gravimetric surveying" [5]. This is a goal which can be split into an engineering part and a scientific part; the engineering part is to develop a concept for a rover and develop a suitable gravimeter. Specifically, Canadensys is developing the VEGA space gravimeter for surface gravimetry on planetary surfaces and asteroids. Due to this recent development, the only information on this gravimeter used in this report is its expected accuracy of 0.1 to 1 mGal (10^{-6} to $10^{-5} m/s^2$). The scientific part of this goal reflects strategies for surface exploration, the number of gravity measurements needed, expected results and complications, noise identification and cancellation and data handling. An important detail is that there is no actual data of lunar gravity surveys in this context available. Only one experiment with lunar surface gravimetry has ever been performed in the Apollo 17 mission, with a different goal and accuracy than this concept. Thus, results will be mostly obtained from modelling and simulations.

The main goal of this preliminary surveying on the surface would be to investigate what kind of cave is underneath the lunar skylight. While an ordinary meteorite crater is very unlikely in this situation, a small, spherical void could be underneath the hole, a result of a localised collapse. This would still be of scientific interest, but sending a probe there for

a two-week exploration mission would be a waste. The aim of the mission would specifically be to prove the existence of the previously mentioned lava tubes and gain insight in how these formed and what their morphology is. The idea is that with non-invasive gravity surveying, this difference can easily be spotted. A mission constraint also becomes relevant here, which is the maximum depth a rover could go to. While no concrete details have been specified regarding the cable length for a descending rover, the maximum length has been loosely set at about 150 meter. Caves can be found much deeper than that, and these are also detectable. Ideally, the gravity survey also gives an answer how deep the cave is which is providing the gravity data.

4.3. GRAVITY SURVEYING

The power of gravity measurements to get to know a subsurface has been known for quite some time, and since the first half of the twentieth century, this has been used in field applications. However, due to small density differences and the small scale of most subsurface objects or rock layers, interpretation of data was often still too large of a challenge. These issues are much less of an issue with cavity detection, as the density difference is above all other natural phenomena, and due to the origin of caves (see Section 4.1 for more), caves are usually very extended. Already in 1963, this principle was used to detect caves, for example by Colley [2]. Many researches later have used similar techniques on different locations for subsurface probing ([19], [20], [21], [22]). These are all terrestrial applications, but the principles work equally well on the lunar surface. Despite the power of this simple gravity survey, limitations also become clear quickly when looking at two figures from Colley [2], shown in Figure 4.2 and Figure 4.3. The first figure shows how a cylinder's size must increase with depth in order to be measurable; all cylinders above the curve are measurable, all beneath are not. The second figure shows nicely the difference between a shallow and a deep cylinder on the gravity signal. While they are distinguishable, the deep cylinder profile is difficult to identify due to the presence of a regional curve. Other surveys or theoretical predictions must be performed in order to correct for this, which is the topic of Section 4.4.

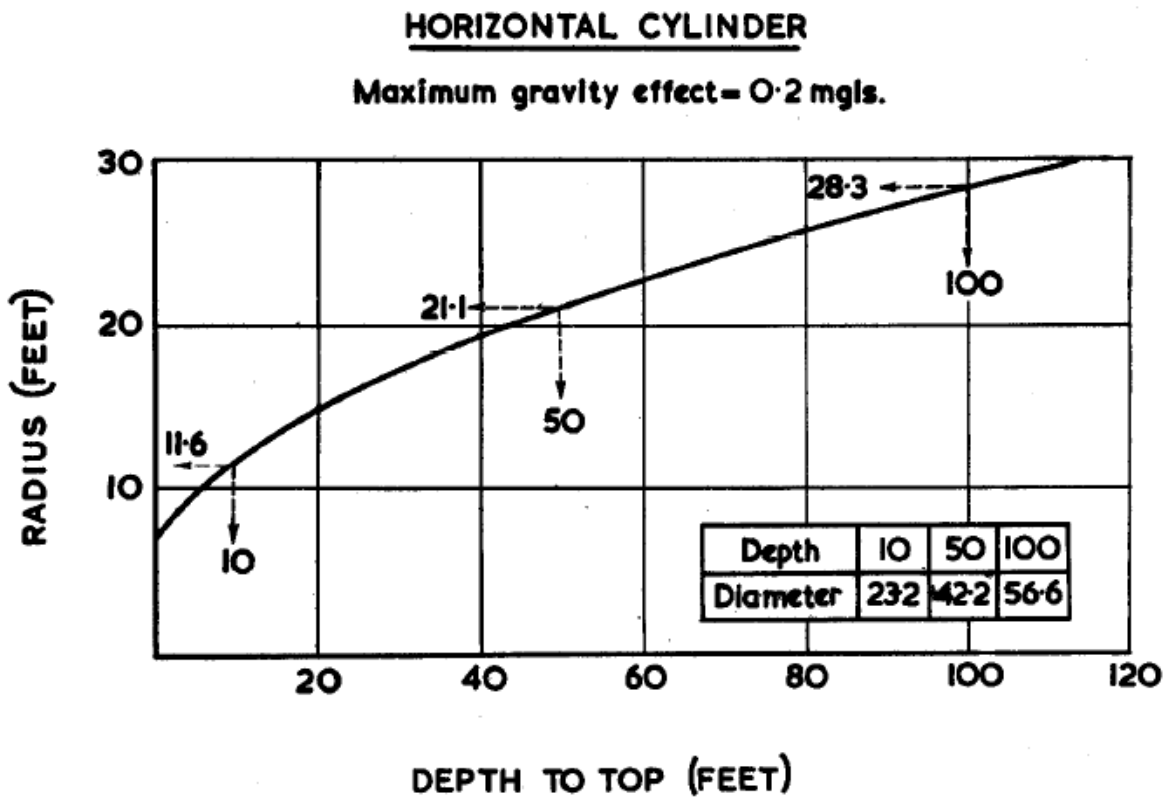


Figure 4.2: Minimum size of a buried cylinder for detection on the surface, assuming an accuracy of 0.2 mGal, units in feet [2].

Before Colley's research on gravity surveys for cavity detection, gravity measurements for exploration of geophysics didn't catch much attention [2]. Since then, numerous reasons outside of purely scientific interest have been found for gravity surveys for cavity detection. There is much interest in the petroleum industry, as this technique can be used for shallow oil and has reserve detection. Similarly, in dry desert areas, water reserves could be found this way. Regarding defence, candidates for shelters and bunkers can be found this way, as well as emergency storage. With caves also being a large tourist attraction, gravity surveys can also be used to find caves for commercial exploitation. Another very important reason for such research in some areas is given by Saibi *et al.* [3], namely structural stability of the underground, or detection of 'geo-hazards'. Due to human industry or geophysical processes, the stability of the surface can decrease to

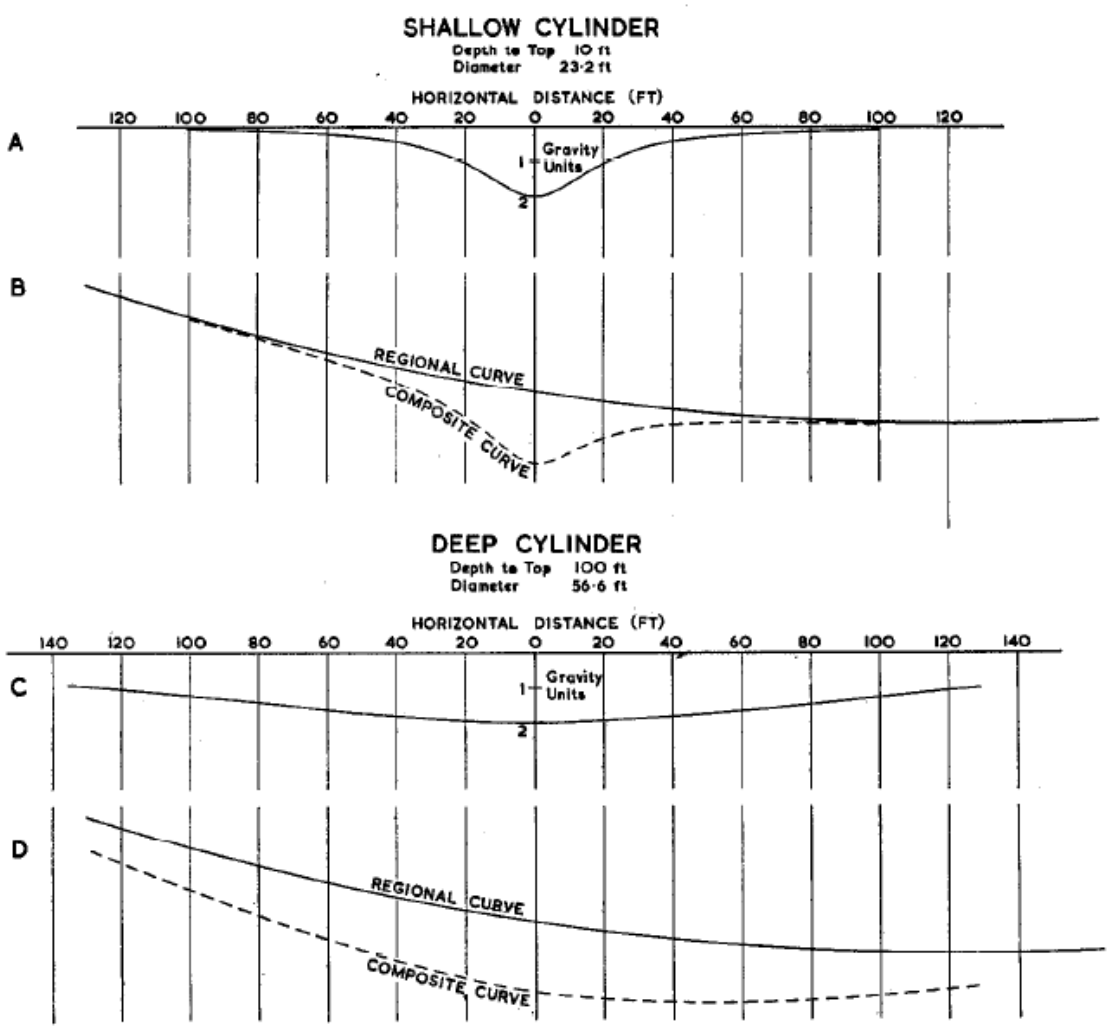


Figure 4.3: Effect of a buried cylinder on the gravity signal, with a regional natural gravity gradient, units in feet [2].

dramatic levels. The sudden appearance of sinkholes, where the surface simply collapses, are a sign of that. Of course, one would want to get insight in the moment such disasters can happen, and perhaps evacuate buildings or areas before that.

Scientific reasons for these gravity surveys can be found in archaeology, speleology and geology, with for example understanding how these cavities formed, whether they show any new, unknown information and what their morphology is. Regarding the lunar lava tubes study, ESA's stated goal is to get more information on possible cave structure, size, firmness, stability and origin, but also all in the context of possible future human bases, as these cavities could provide excellent protection against radiation and meteorites, while also being very stable in for example thermal conditions [18].

While gravity surveys are a promising way to get to know the interior, just the data alone is not useful. Corrections need to be applied to these data, as the raw data is noisy and scattered with anomalies and trends. The next section, Section 4.4, discusses these corrections

4.4. Corrections and errors

There are three main categories of corrections; geological corrections due to the presence of certain structures, interpolation errors due to a finite number of measurements, and other, various corrections. All these corrections introduce errors, meaning that the total error or accuracy of the gravity map depends on dozens of factors. As the VEGA accelerometer is still in development by a private company, specifications on how they calculate their intended accuracy of 0.1-1 mGal on the Moon are unknown. However, as they concern the measurement alone, thus the technical parts of the accelerometer, it will be listed as a separate error in the 'other' category.

When one obtains a discrete gravity map through gravity surveying, there are two steps to take first. One of them is some kind of interpolation, if needed. As there are only a finite number of measurements taken (for the concept mission, this number is estimated to be thirty to fifty) and a map is supposed to be continuous, at some parts of the map, measurement values are missing. The usual way to account for this is to use the measurement values which are present to estimate

the gravity values on other places. Numerous methods exist for that, and it is impossible to quantify the error which is made correctly here, but in general the error is larger the fewer measurements there are and the more complex (i.e. large gradients, few predictability) the gravity signal is. Thus, the number of measurements should be suited to the expected signal. An example of how interpolation can lead to errors is shown in Figure 4.4 below, obtained from the paper by Saibi *et al.* [3]. In (a), the actual map is given of the resulting gravity by a large number of buried spheres. The black squares are the random measurements, clarified in (b). Based on this, an inversion algorithm could end up with the situation at (c), which is similar to the 'true' map of (a), but with locally many errors. Only five or six clear anomalies are distinguishable, for example, while there are dozens present. Note that these structures are relatively small and all very similar; in the lunar mission concept, the target is a very large cavity. Still, the interpolation error is important and partially preventable. It pays of to do a short study regarding interpolation in situations with realistic cavities, for example to find the optimal gravity station pattern, or a suitable number of gravity stations.

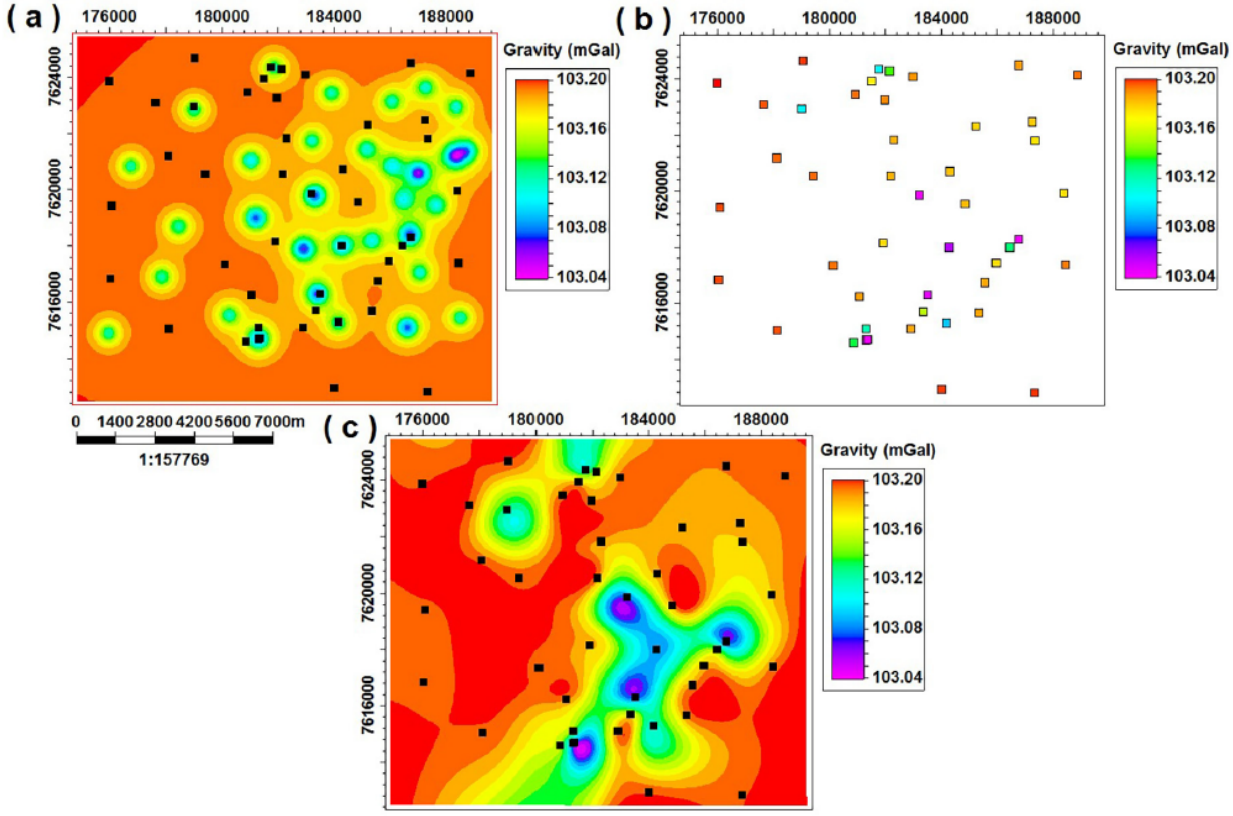


Figure 4.4: Forward modelling gravity results on the whole grid (a) and only on stations (b). (c) shows interpolation of data from (b) [3].

With the (interpolated) map, one can start the interpretation. The map already shows local differences and perhaps a regional pattern, but for understanding what is present in the sub-surface, in general the so-called *Bouguer anomaly* is used, the difference between the observed gravity value and the theoretical gravity value caused by an homogeneous Earth. Unless the subsurface geological layers are known to a very good accuracy, it is not recommended to take these into account at this step, even if it is known that the subsurface is certainly not homogeneous in terms of density [23]. The formulae for the Bouguer anomaly and the theoretical gravity are as follows:

$$\begin{aligned} \Delta g &= g_{obs} - g_{th} \\ g_{th} &= \gamma + \delta g_h + \delta g_{bpl} - \delta g_{ter} \end{aligned} \tag{4.1}$$

Here Δg is the theoretical gravity value, g_{obs} is the observed gravity value from the gravity map, g_{th} is the theoretical gravity value, γ is the *normal* gravity, δg_h is the *free air reduction*, δg_{bpr} is the *Bouguer plate reduction* and δg_{ter} is the *terrain reduction*, all in mGal. The latter four terms are discussed later. Note that the Bouguer anomaly Δg is what is usually used for interpretation, as this shows all gravity anomalies and inhomogeneities, devoid of any global or regional predictable trend [23].

Next, the individual corrections are listed:

- Normal gravity reduction γ; the expected gravity acceleration on a certain location of a body like a spheroid, ellipsoid. The geoid model can taken for the gravity acceleration, or any shape defining the Earth or Moon, as well as the latitude to also include the centrifugal acceleration. Rather than assuming the density of the Earth, it is more common to calibrate this value according to gravity values measured on other parts of the body, such as the equator.

For both the Earth and the Moon, plenty of calibration data exists. Note that for all such corrections, a 'Flat-Moon' model is used to simplify equations, leading to a very small but negligible error.

- Free air reduction g_h; when the gravity station is above the reference level, a free air reduction must be applied. This is since there is now also air or vacuum between the measurement point and the body of interest. If the Moon would be perfectly spherical, the following formulae would hold:

$$\begin{aligned} g_m &= \frac{G \cdot M}{r^2} \\ \frac{\partial g_m}{\partial r} &= \frac{-2 \cdot G \cdot M}{r^3} = \frac{-2 \cdot g_m}{r} \end{aligned} \tag{4.2}$$

  g_m is the gravity acceleration on the lunar surface, G is the universal gravitational constant, M is the mass of the Moon and r is the distance between the gravity sensor and the centre of the Moon. While an approximation, the value of 0.19 mGal/m on the Moon is commonly used for the free-air correction [24]. This means a gravity value decreases in magnitude by 0.19 mGal for every meter you are above the surface, which is the assumed shape of the body (for example the geoid or perfect ellipsoid).

- Bouguer plate reduction g_{bpr}; this correction is similar to the free air reduction, but regarding rock masses. Again comparing to the reference level of the assumed shape for γ, a Bouguer plate reduction is applied when there are masses between the measurement station and the reference level. While for the free-air correction, a cubic meter of terrestrial or lunar atmosphere was considered mass-less, this can't hold for the stone. With a sufficiently accurate shape model, Bouguer plate reductions are usually in the from of flat plates, leading to the following correction formula:

$$\delta g_{bpl} = 2 \cdot \pi \cdot G \cdot h \cdot \rho \tag{4.3}$$

  Here h is the height and ρ is the density of the rock between the reference level and the gravity station.

- Terrain reduction; lastly, there is terrain reduction. The terrain is usually not included in shape models, and both nearby hills or plateaus as well as nearby valleys will lead to a local decrease in gravity acceleration magnitude. However, when a certain shape is assumed for them, the gravity signal caused by these mass surplusses or mass deficits can be calculated and corrected for. When the terrain reduction has been applied, little correlation between the terrain and the Bouguer anomaly map is expected.

The resulting Bouguer map should show only anomalies caused by subsurface densities, different than modelled. This can be anything, ranging from ores, rock layers, but also cavities and full caves. It will be impossible to retrace every small anomaly to an object, as many will be overlapping and interpolation is still occurring, but the expectation is that a lava tube would leave a large and easily identifiable trace in the Bouguer map. The other anomalies can be considered noise, which would always be present.

Next follows the list of 'other' corrections and errors; some of them can be accounted for, some can't:

- As the VEGA accelerometer is an absolute accelerometer, it determines the magnitude of the total acceleration. This is mostly determined by the Moon, as this is the closest large body when on the surface of the Moon. Yet, due to their size, the Earth and the Sun also have a influence which can perhaps be detectable. Using the approximate formula $g = \frac{GM}{r^2}$, for M the masses of the Moon, Earth and the Sun and for r the Moon's radius, the Earth-moon distance and the closest Sun-Moon distance, the following values are obtained:
  - Moon: $1.624 \cdot 10^5$ mGal
  - Earth: $2.697 \cdot 10^2$ mGal
  - Sun: $1.004 \cdot 10^1$ mGal

  With an intended accuracy of 0.1 to 1 mGal, these are significant influences to take into account, and it is worth finding an accurate formula and values as well as the position of the Earth and the Sun in the lunar sky in order to correct for this. While these corrections will be practically the same for all, for the interpretation of data this correction is needed.

- Very often, only the vertical component of the gravitational acceleration is of interest, denoted by g_z. This direction is inferred from the orientation if the instrument. If the instrument is inclined compared to the actual direction of g_z, a different value will be measured. An inclination of θ results in a measurement of $g_z \cdot \cos(\theta)$ instead of g_z, and for small angles this error can be approximated by $e_\theta = 0.5\theta^2$ [24]. Thus, the accelerometer is accompanied by gimbals to make sure the instrumentation is as perpendicular to g_z as possible. Still errors persist; for the Apollo 17 gravity experiment, this error was at maximum $\theta = 00°03'$, for example, leading to an error of 0.06 mGal at most [24]. Especially with robotic instrumentation, this is important to keep note of.

- A big advantage of gravity surveys on the Moon compared to Earth is the number of temporal variations of the gravity field. There is no wind, human activity or abrupt movement present on the Moon, in general. However, just like the Moon creates tides on the Earth, the Earth also generates tides on the Moon, or rather in the Moon. There are solid tides in the interior of the Moon and the Earth, which can vary as much as 0.29 mGal on Earth [23]. These values come from theoretical tidal predictions, as certainly for the Moon, no tidal force measurements are available for every location. Thus, while corrections can be done, (significant) errors are made once again here.

- There are also two random effects leading to movement which can be present on the Moon, namely 'moonquakes' and thermal effects. Quakes in the interior and on the surface of the Moon are abundant, and it is known that earthquakes can disrupt gravity measurements [8]. Most moonquakes are caused by solid tides in the interior, impacting meteorites and the sudden heating of the surface crust after a lunar night. Not only the surface has thermal effects; the rover or instrument can also have its own thermal effects. For the Apollo 17 gravity instrument, it is known that it was extremely sensitive to temperature, and it needed to be regulated such that the temperature was stable up to 0.01 K [24]. An oven/fridge was used to regulate the temperature. These requirements can differ per instrument, but in any case the temperature needs to be either well known or very stable. It is not so much a permanent gravity difference caused by these effects that disrupts measurements, but more vibrations and shocks in the instrument itself. These can't be corrected for, and need to be avoided largely. This is also a reason why the mission concept has one or two days available for calibration after sunrise on the Moon.

- Lastly, there are internal errors, uncertainties and inaccuracies. Degrading spring tension for an accelerometer based on springs for example is an effect which can create an error in the measurements. These internal effects and technical details are the responsibility of the manufacturer and designer, and the value 0.1-1.0 mGal for the total accuracy is assumed.

Due to all these effects, some of which could be temporal, it is in general advised to measure at a reference point every two hours (on Earth) for validation [23].

4.5. Artificial Neural Networks

For many inventions in engineering and science, we have been inspired by nature; this is called *mimicry*. By looking into nature, we managed to both imitate it and learn more about it. Billions of years of evolution have lead to biological structures and mechanisms work so efficiently, that often humans can only imitate this to some extent. However, being inspired by nature already gave us crucial inventions in fields of flights, aerodynamics, solar energy and locomotion. Apart from these rather physical inventions, there is also much to learn about intelligence, a matter which still provides us with many mysteries. State-of-the-art robots have great flexibility, but lack the navigation and decision-making tools that small insects such as fruit flies have, despite their small brains or brain-like structures. Especially the field of machine learning and artificial intelligence has much to learn from nature, as its goal is to imitate this whole concept of intelligence.

One way of imitation is to use a simple model of how a brain works, on its lowest levels. The building blocks of our brains are cells called *neurons*, which continuously fire signals to neighbouring cells. They are built with many appendices to optimise the signal transition network. While we are unsure of how these neurons exactly work, one of the common interpretations is that a concept or thought is only present in a group of neurons; individual neurons are just intermediate messengers. This interpretation implies that neurons have little significance on their own, and the power is in a network of neurons. Thus, neurons would simply receive and transmit signals, which can be imitated by simulations as well. In its simplest form, a neuron can be considered an object which combines inputs into one output, which will be the input for another neuron. Arriving at this kind of approach, we enter the field of 'artificial neural networks' (ANN's); networks built of objects considered artificial neurons, imitating a brain-like structure in its simplest form.

Regarding these ANN's, a common simple structure is a *feed-forward neural network*; here the neurons are divided into individual layers, and each neuron in each layer only has connections with neurons in the next or the previous layer. More specifically, a signal starts at the first layer and is fed as input for the next layer, where operations are done on the input to transform it to an output, serving as an input for the next layer. Signals never are transmitted back in the network, leading to a strictly forward-feeding of information, without cycles. Each neuron in a particular layer is connected with each neuron from the next layer, and every one of these connections has a specific *weight* associated with it. This weight is multiplied with the transferred signal in order to have more control on it. A similar thing happens when combining all incoming signals to a particular neuron; a *bias* is added as an additive constant for more control. Thus, every neuron in a system has an individual bias, and every connection between neurons has an individual weights. These biases and weights are what makes a neural network unique; by changing them you can obtain any output from any input. The trick is to adjust these weights and biases according to training and testing data fed to the network.

Neural networks mostly see applications in pattern or object detection, optimisation or data interpretation; in general applications which need to be repeated often and need to be automated. In this specific research, the application is the previously mentioned gravity data interpretation. Interpretation of gravity measurement data requires quite some exper-

tise and precise work, which currently can't be automated. Existing methods include mathematical inversion techniques in variant ways; these are useful and robust, but require much computational power and usually result in density 'smears', which are not representative of a cavity. The location and orientation of the cavity can be obtained this way, but not the depth and shape. Neural networks can help resolve this issue.

Of course, due to the simple structure of these neural networks, they are not expected to outperform existing methods, but they can provide unique advantages. For example, with constraints on the shape of the cavity, interesting results can be obtained. The research done by Salem *et al.* [7] gives a nice view on this. They constrain the shapes to consider to the following options:

- A perfect sphere
- A semi-infinite vertical cylinder
- An infinite horizontal cylinder

Fortunately, these basic shapes are very good approximations for the shapes we expect to encounter on the Moon, looking for subsurface lava tubes. As mentioned previously, lava tubes are expected to be usually very elongated, with a radius perhaps up to several kilometers, but a lateral size of several tens of kilometers. As the extent of a gravity survey on the Moon by an assumed rover will at most cover several hundreds of meters, a subsurface horizontal cavity will appear infinite in length. The skylights on the other hand, which offer an entrance to the buried lava tube, are almost perfectly vertical, as they are the result of a collapse of material into the lava tube. While here the semi-infinite length falters, it still is expected to be easily discernible from other shapes. Here it holds that the less good this approximation is, the more fortunate this is, as a skylight limited in length provides easier access to the lava tube. Lastly, there is the sphere, a less accurate representation of a subsurface lava tube, but also a possible shape for a subsurface cavity. Spherical cavities can be the result of lava upwellings, totally collapsed lava tubes, or other natural processes. In any case, exploration of a spherical cavity is of much lower interest than exploration of a 'proper' lava tube. Going back to one of the initial goals of the project, discerning the difference between a subsurface sphere and a subsurface tube, this categorisation is a great fit. Salem *et al.* [7] provides us with a general formula for the gravity signals of these shapes:

$$g(x) = \frac{Az^m}{(x^2+z^2)^q} \tag{4.4}$$

Here, g is the vertical component of the gravitational acceleration in m/s^2, x is the shortest distance between the measurement position and the maximal anomaly in m (in the case of a horizontal cylinder, the maximal anomaly is represented by a line), z is the vertical distance between the measurement position and the centre of the cavity in m (the depth). The other symbols are shorthand notation; A is an amplitude factor in m^2/s^2, m a dimensionless exponent and q the dimensionless *shape factor*. They are given by the following formulae:

$$A = \begin{cases} \frac{4\pi G\rho R^3}{3} & \text{for a sphere} \\ 2\pi G\rho R^2 & \text{for a horizontal cylinder} \\ \pi G\rho R^2 & \text{for a vertical cylinder} \end{cases}$$

$$m = \begin{cases} 1 & \text{for a sphere} \\ 1 & \text{for a horizontal cylinder} \\ 0 & \text{for a vertical cylinder} \end{cases}$$

$$q = \begin{cases} 3/2 & \text{for a sphere} \\ 1 & \text{for a horizontal cylinder} \\ 1/2 & \text{for a vertical cylinder} \end{cases}$$

While this formula seems useful for a generalisation between shapes, there appear to be still issues, most importantly that there are still four unknowns; given a measurement g(x) and a horizontal x, still the shape factor q, depth z, radius R and density difference ρ are unknown, and due to the nature of the problem, given the known g(x) and x, an infinite number of combinations of the other parameters can be given that either fit the measurements completely, or almost. This can partially be circumvented by supplying more data, but with the presence of noise and the fact that many approximations are used, the problem is not solved.

Salem *et al.* [7] proposes to normalise the observed gravity measurements. For this, the maximum gravity anomaly is needed, which can be found at $x = 0$ (either interpolated or observed). The result is noted down as:

$$g_0 = g(x=0) = \frac{Az^m}{(0^2+z^2)^q} = \frac{Az^m}{z^{2q}} \tag{4.5}$$

Now, every measurement is divided by this value to normalise it:

$$g_n(x) = \frac{g(x)}{g_0} = \left(\frac{z^2}{x^2+z^2}\right)^q \tag{4.6}$$

As can be seen, $g_n(x)$ only depends on z, x and q, and x is considered a known variable after measuring. The other variables R and ρ have been filtered out, yet with an estimation of z, q and either R or ρ, the other can be estimated. For the purposes of this research, it is the most interesting to know the depth z and the shape factor q to see whether the cave is worth exploring. Lastly, Salem *et al.* [7] gives the formula for z if q is either known or assumed:

$$z = \sqrt{\frac{\sum_{i=1}^{N}(1-(g_n(x_i))^{1/q})(g_n(x_i))^{1/q}x_i^2}{\sum_{i=1}^{N}(1-(g_n(x_1))^{1/q})^2}} \tag{4.7}$$

Here, N is the number of measurements, the other parameters are as mentioned before. The implication is that with one 'perfect' measurement and assuming q, z can already be obtained, and with two 'perfect' measurements both z and q can be obtained, which is very efficient. With 'perfect', it is meant that the reality corresponds to the approximation, and no noise is present. Of course, this will never be reality, and in the end result, one would use some kind of least-squares estimation or averaging, such as Salem *et al.* [7] proposed for the 1D-variant, looking for z. With many observations, noisy data and cavity shapes which do not correspond to the assumed shapes, this approach again leads to much computation. An alternative idea, proposed by many authors in this field, is the use of neural networks. As input, one provides a tuple of a gravity measurement and the horizontal distance to the maximum anomaly, and as an output one wants one or two parameters out of z and q. The neural network imitates the structure of the approach of looking for z and q, without having an understanding of what needs to be done. The result is a long 'training time', in which the neural network tries to understand the structure of generated data, but when the neural network has this structure stored, application to real-life data is extremely fast. The idea is that the accuracy of the result matches with the accuracy of the neural network applied to the training or testing data.

4.6. Neural network architecture

For the set-up of the problem, a feed-forward neural network was initially chosen with two input neurons, one intermediate or *hidden* layer of neurons, and an output layer with two neurons. In order to gain more insight in the inner workings of a neural network, no framework was used, and the neural network has been programmed from scratch, without existing packages. Due to this also, the initial network is relatively simple. However, results from other authors have showed that simple structures can work perfectly for this type of application [6] [25] [8]. On a similar note, with only one hidden layer, any desired output can be simulated with enough hidden neurons, following the theorem by Cybenko (1989) [26].

The usual steps of using an ANN are laid out in Table 4.1 [4]. This general flow is also used in this project

Table 4.1: Nine steps in designing an ANN estimator model [4]

Step 1	Variable selection
Step 2	Data collection and processing (preparing training data)
Step 3	Assemble training and test data
Step 4	ANN paradigm
	Number of hidden layers
	Number of hidden neurons
	Number of output neurons
	Transfer function
Step 5	Evaluation criteria
Step 6	ANN training, Number of iterations for training
Step 7	Testing ANN with new synthetic inputs without noise
Step 8	Testing ANN with new synthetic inputs with noise
Step 9	Testing ANN with new real data

Regarding the training, the standard back-propagation with a gradient descent method is used, which is explained later in Subsection 4.6.4. However, rather than a supervised neural network, where the correct output is provided, the neural

network is self-supervised, using its own output and input to judge its error. This is to avoid having to label data, to make the neural network more flexible, and to learn how to go a step beyond the usual level of neural network development. Comparisons with supervised neural networks are also done. Future plans include usage of more advanced, existing neural network structures for better performance.

4.6.1. Data and normalisation

A crucial part of neural network development and usage is the availability of training and testing data, from which the neural network learns what to do. It is crucial that this data is representative of the ultimate application setting, as well as being large in size. In general, the more data is available, the better the performance of the neural network will be, though the training will also take longer [27] [4]. There are several types of data used in this project, listed below:

- First, based on the theoretical formulae for gravity anomalies caused by spheres and cylinders, parameters can be randomly generated and with these parameters, tuples of gravity measurements and distances x can be obtained. This is done with the use of formula 4.6; q and z are assumed for a particular cavity, and the goal is to let the ANN estimate these parameters q and z only with the input of x and g_n. The advantage of this method is that no gravity modelling software is required, it is quick and it is easy, additionally as much data can be generated as desired. However, this data is not yet representative of the situation.
- When the neural network is able to work well with the previous data, gravity modelling programs are used for actual cavity modelling. This includes software like IGMAS and the semi-infinite spectral element simulation method by Gharti *et al.* [28]. The difference with the previous 'perfect' data is that in this modelling part, the shapes are not (semi-)infinite and not perfectly round or spherical, making the obtained data more realistic, yet also more computationally intensive to generate. Yet, the situation can still be made more realistic.
- For the next data batch, artificial noise is added to the shapes, as well as random density anomalies in the terrain. At this point, one is modelling the whole system of rock and cavity, generating a signal which resembles what one would actually measure on the Moon. The gravity anomaly is then respective to the gravity signal generated by rock without a cavity underneath, making it more difficult to obtain the values specifically for the cavity. The shapes of the cavity are still the usual spheres and cylinder, however.
- When the neural network is able to also work with this kind of data, one can start to also use terrain elevation and variation, different stone layers with different densities, and geometric shapes different than the ones previosuly mentioned. However, the formulae stay the same, meaning that the task of the neural network becomes to judge whether, say a prism, more resembles a sphere or a cylinder. With the complexity of the problem becoming larger, the goal of the neural network shifts more to distinguishing between a sphere and a horizontal cylinder, and estimating the depth of the cavity.
- If this also succeeds, terrestrial lava caves can be used as lava tube models, adjusted to lunar conditions and possibly with a vertical skylight present. With all these additions, the aim is to have the model be a realistic representation of lunar conditions and investigate how well the neural network would work in-situ.
- As a last, optional step, the neural network could be tested on actual measurements on Earth of subsurface cavities or on the measurements done by the GRAIL spacecrafts on the Moon. With much research done on these topics, the results of the neural network can be compared with actual measurements or more advanced estimations. However, it is expected that more research is needed before this neural network would perform well on such data

It is important to note that, while the neural network is in principle self-supervised and works on its own, it both still estimates its own error, and in most of these cases, the actual shape factor q and cavity depth z are still known and can be used to judge the performance of the neural network. Also, due to the limited scope of this project and the complications which can occur on the way, it is not expected that all these steps will be completed as desired.

Lastly, it is usually advised to normalise the input data, sometimes also the output data [4] [27]. This ensures that values will not 'explode' during multiplication in the neural network, and in general makes the neural network more robust. One of the pillars of the idea of this neural network project is that g_n is normalised, which leads to the decision to also normalise the x-coordinate input.

4.6.2. Neurons and layers

As mentioned before, the theorem by Cybenko (1989) ensures that a single hidden layer can be enough to make a neural network work as desired [26], if enough hidden neurons are present. However, more layers may add needed complexity to a network, outperforming networks with a similar number of neurons in one layer. Yet, the general practise when developing neural networks is to increase the number of hidden neurons rather than the number of hidden layers [4].

Usually, one or two hidden layers is the advised number, certainly for applications in gravity data interpretation [4] [8]. However, also in this hidden layer or these hidden layers, a small number of hidden neurons is advised; numbers between five and twenty are common [4].

For the input layer, there are two general approaches regarding the number of input neurons to use for gravity data interpretation. The first states that as a tuple of a gravity measurement and a distance to the maximum anomaly combines into one measurement, this should be the input only. This results in two input neurons with values being this gravity measurement and the distance. However, as mentioned previously, one tuple is not enough to get both the shape factor q and the cavity depth z from the problem equation; the problem is under-constrained. Thus, another approach is to use all gravity measurements of one survey as the input for the neural network at the same time. For example, when the data of 25 gravity survey stations are used, 25x2=50 input neurons are used. Yet, there are also limitations with this approach. It is not immediately clear how the network should be used if a gravity survey consists of more or less than 25 measurements. Also, it should be investigated if the order in which the measurements are provided to the neural network influences the results, as this should not be the case. In this research, both methods will be used to look at the optimal performance.

Regarding the output layer, the usual practice is that the value(s) of the output neuron(s) are also the output of the neural network on itself. While some authors in this field, such as Eshaghzadeh and Hajian [9], have used four output parameters to get the shape, depth, radius and density difference, in this project the approach of Salem *et al.* [7] is used, based on earlier works of [6]. This means that the radius and density difference are not considered important for the problem, and one can be estimated manually if the value of the other is somewhat known. This leads to two important parameters; the shape factor q and the cavity depth z. The initial idea is then to use these parameters as output variables for the neural networks, leading to two output neurons. However, due to the limited number of possible shape factors (only three values), an even simpler approach can be used, where the shape factor is assumed and only the cavity depth is sought for. Three different neural networks can be made for this, though this makes it more difficult to obtain the most likely cavity shape from the measurements.

4.6.3. Inner workings of a neuron

Each neuron combines the input information from the previous layer (the input neurons receive this from the input data) by multiplying the values of the neurons from the previous layer by the weights connecting the neurons. In this way, via the weights all neurons from the previous layer have their own influence on the value of a neuron in the current layer. All these neuron value-weight combinations are added as the total input value for the current neuron. This gets added to a bias, unique to the neuron, and based on this the neuron sets a value for itself as well. This process of going from an input to an output is governed by an *activation function*, which can be any function mapping input to output. However, as this input-output conversion is an important part of the process of making the neural network behave as desired, it is crucial to pick an appropriate function. A desired property for example is that the function is differentiable and continuous, such that gradients can be computed and used. Furthermore, it can be desirable to have a clear distinction between positive feedback and negative feedback in the activation function to simulate strong positive or negative feedback from inputs. With the weights, the relative influence from certain inputs or neurons can also be adjusted.

A visualisation of what is happening can be seen in Figure 4.5. By smart adjustment of the biases and weights of each neuron and neuron connection, a combination of simple ReLU activation functions can be combined to simulate a sine wave. Here, many weights are still absent, and with more weights and neurons the simulation can be even more realistic.

Common activation functions are [4]:

- Linear: $y = x$ This is the most straightforward activation function, as the input is the same as the output. With this function combined with weights and biases, many functionalities can be replicated, though the behaviour of the function is global rather than local due to the function having no limits.
- Heavyside: $y = 0$ for $x < 0$, $y = 1$ for $x \geq 0$. Another very straightforward function with only two outputs values. the result is bounded and it operates like a logic gate; there either is an output signal or there is not. Useful for adjusting local behaviour, but the zero derivative everywhere gives issues when adjusting weights and biases. Despite this, it can be used in simple applications.
- ReLu: $y = 0$ for $x < 0$, $y = x$ for $x \geq 0$. This function is an abbreviation of 'rectified linear' function, and it is capped for low input values. Thus, there is only output when the input is reasonably high, which can be adjusted with weights and biases. Due to this, the ReLU activation function is suitable for adjusting local behaviour with some neurons, while having a global nature for others.
- Sigmoid: $y = \frac{e^x}{e^x+1}$. This function involves more computation, but is continuous and differentiable, and capped at $y = 0$ and $y = 1$. These values can be changed by changing weights and biases, but it shows that the range is always bounded, making this activation function useful for simulating local behaviour.

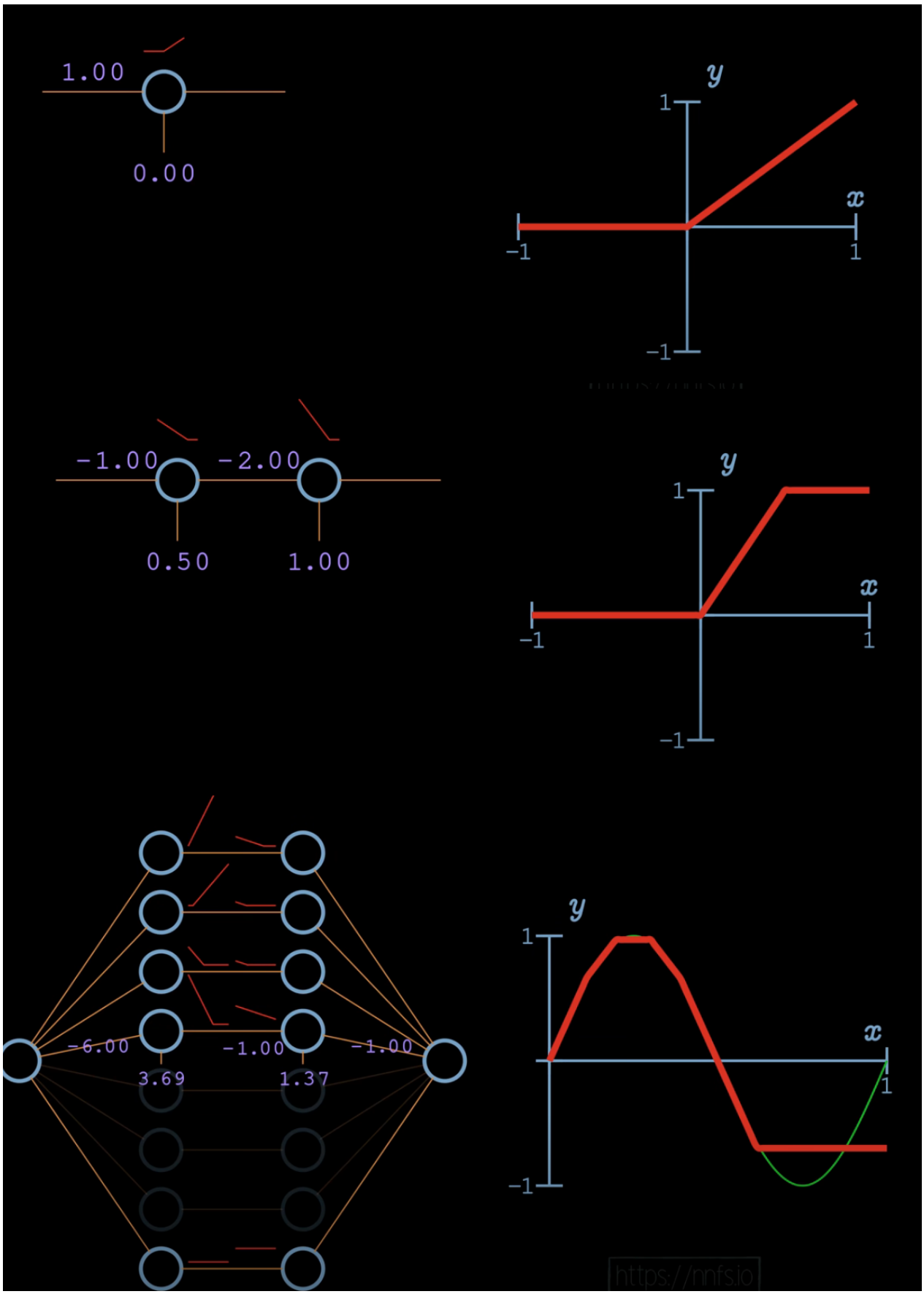


Figure 4.5: A visualisation of what activation functions can do in combination with weights and biases (from: Python web series by sentdex)

While in principle every neuron can have its own activation function, this option usually merely adds complexity without providing much in return [4]. Activation functions are usually chosen per layer, where it is usual to us a linear activation

function for both the input and output layer. In this project, only the few main ones will be used, as they have shown to work well in similar researches, for example by Elawadi *et al.* [25], Hajian *et al.* [8], Hajian and Styles [4] and Eshaghzadeh and Hajian [9].

4.6.4. BACK-PROPAGATION

So far, the neural network merely consists of a network of neurons, weights and biases. An input can be fed to the network, and via the neuron values, connection weights, unique biases and the activation functions, the output of the neural network can be calculated. At initialisation, these weights and biases are usually randomly generated. Thus, the first output will be random as well, if training has not occurred yet. The training is the process where these weights and biases are updated according to some kind of error. In the most straightforward form, the error is the difference between the actual output and the desired output, if known. This is an example of *supervised learning*; the data needs to be labelled in some way, such that the neural network can learn from the correct answers. Some projects, such as this one, also lend themselves well for another approach, where a neural network can estimate the error on its own. With this method, no human intervention is needed, and the whole process of data acquisition to neural network training and application can in principle be automated.

While no previous explicit literature on self-supervised learning for neural networks in gravity data interpretation has been found known to the author, with the works of Salem *et al.* [7] it is relatively straightforward. First, the usual formula for a single input is presented:

$$E = \frac{1}{2}\sum_{i=1}^{n} e_i = \frac{1}{2}\sum_{i=1}^{n}(d_i - y_i) \tag{4.8}$$

Here E is the total error or 'energy', n is the number of output variables, e_i is the individual error, d_i is the desired output and y_i is the actual output. This is the formula for supervised learning, where the desired output is known. For the network with output variables z and q, this becomes the following:

$$E = \frac{1}{2}\left(e_z + e_q\right) = \frac{1}{2}\left((z_{actual} - z_{NN}) + (q_{actual} - q_{NN})\right) \tag{4.9}$$

One way of adjusting weights and biases during training is to look how the total error depends on these weights and biases in terms of the gradient. Back-propagation is based on this principle, which is one of the simplest learning mechanisms. For this to work, one starts with the derivative of the total energy to the output of the output layer; one is interested in $\frac{\partial E}{\partial z_{NN}}$ and $\frac{\partial E}{\partial q_{NN}}$:

$$\begin{aligned}\frac{\partial E}{\partial z_{NN}} &= \frac{\partial E}{\partial e_z}\frac{\partial e_z}{\partial y_z} = e_z \cdot -1\\ \frac{\partial E}{\partial q_{NN}} &= \frac{\partial E}{\partial e_q}\frac{\partial e_q}{\partial y_q} = e_q \cdot -1\end{aligned} \tag{4.10}$$

However, using formula 4.4, another option is possible:

$$E = \left(\frac{z_{NN}^2}{x^2 + z_{NN}^2}\right)_{NN}^{q} - g_n(x) \tag{4.11}$$

For a neural network structure with more than two input neurons, this formula can be changed to a sum of errors. Based on the outputs of the neural network, a prediction can be made about the gravity signal at a distance x, if the output variables z_{NN} and q_{NN} were correct. The difference with the actual gravity signal can then be calculated and used as error. This works, as there is a relatively simple way to connect the output again to the input. The derivatives then become:

$$\begin{aligned}\frac{\partial E}{\partial z_{NN}} &= \frac{2 \cdot q_{NN} \cdot x^2 \left(\frac{z_{NN}^2}{x^2 + z_{NN}^2}\right)^{q_{NN}+1}}{z_{NN}^3}\\ \frac{\partial E}{\partial q_{NN}} &= \left(\frac{z_{NN}^2}{x^2 + z_{NN}^2}\right)^{q_{NN}} \cdot \log\left(\frac{z_{NN}^2}{x^2 + z_{NN}^2}\right)\end{aligned} \tag{4.12}$$

While these formulae seem more complex than the standard, supervised-learning formulae, they can be used equally well, with one note of caution. The gradient values become extremely high when z approaches zero, and are not real when z becomes zero. Of course, such a value, as well as negative $z-$values and $q-$values, are not physical and should be avoided. If the neural network does not do this on its own, measures could be taken which avoid this naturally, such as

initialising biases and weights with only positive values, or weight and bias updates are only performed when this does not lead to such errors in later situations.

The rest of the back-propagation works in a similar way and is relatively straightforward. Weights and biases are changed per layer. For the weights between the last hidden layer and the output layer, we have the following relation:

$$\frac{\partial E}{\partial w_{jk}} = \frac{\partial E}{\partial y_k} \cdot \frac{\partial y_k}{\partial v_k} \cdot \frac{\partial v_k}{\partial w_{jk}} \tag{4.13}$$

Here, E is the total error or energy, y_k is the value of the k'th output neuron (thus, z_{NN} or q_{NN}), v_k is the input of the k'th output neuron (the sum of incoming signals) and w_{jk} is the weight of the connection between the j'th neuron in the hidden layer and the k'th neuron in the output layer. Thus, this relation depicts the relation between the total error and a specific weights, through the dependency on the final output, the relation between input and output and the relation between inputs and weights. These derivatives are all known:

$$\frac{\partial E}{\partial z_{NN}} = \begin{cases} e_z \cdot -1 & \text{for supervised learning} \\ \frac{2 \cdot q_{NN} \cdot x^2 \left(\frac{z_{NN}^2}{x^2+z_{NN}^2}\right)^{q_{NN}+1}}{z_{NN}^3} & \text{for self-supervised learning} \end{cases}$$

$$\frac{\partial E}{\partial q_{NN}} = \begin{cases} e_q \cdot -1 & \text{for supervised learning} \\ \left(\frac{z_{NN}^2}{x^2+z_{NN}^2}\right)^{q_{NN}} \cdot \log\left(\frac{z_{NN}^2}{x^2+z_{NN}^2}\right) & \text{for self-supervised learning} \end{cases}$$

$$\frac{\partial y_k}{\partial v_k} = \frac{\partial \phi_k}{\partial v_k}$$
$$\frac{\partial v_k}{\partial w_{jk}} = y_j$$

Here ϕ_k is the activation function of the neurons in the output layer; $\frac{\partial \phi_k}{\partial v_k}$ is then the derivative of the activation function to the input, evaluated at the neuron value. y_j is the value of the j'th neuron in the last hidden layer. The other symbols have been introduced previously.

For the biases, a very similar formula holds:

$$\frac{\partial E}{\partial b_k} = \frac{\partial E}{\partial y_k} \cdot \frac{\partial y_k}{\partial v_k} \cdot \frac{\partial v_k}{\partial b_k} \tag{4.14}$$

With:

$$\frac{\partial v_k}{\partial b_k} = 1 \tag{4.15}$$

As the bias is simply an additive constant, used only to have a larger influence on the neural network behaviour.

For the other layers, the same formulae hold, but with extra terms, and the additional note that for other weights, there are multiple ways 'leading' to this weight. Thus, the sum of all possible ways has to be computed;

$$\frac{\partial E}{\partial w_{ij}} = \sum_k \frac{\partial E}{\partial y_k} \frac{\partial y_k}{\partial v_k} \frac{\partial v_k}{\partial v_j} \frac{\partial y_j}{\partial v_j} \frac{\partial v_j}{\partial w_{ij}} \tag{4.16}$$

Here, the new derivatives are as follows:

$$\begin{aligned} \frac{\partial v_k}{\partial v_j} &= w_{jk} \\ \frac{\partial y_j}{\partial v_j} &= \frac{\partial \phi_j}{\partial v_j} \\ \frac{\partial v_j}{\partial w_{ij}} &= y_i \end{aligned} \tag{4.17}$$

As usual. k denotes something of the output layer, j something of the last hidden layer and i something of the layer before the last hidden layer. y is an output, v is an input, b is a bias and w is a weight.

For completeness, this formula is also provided for the biases:

$$\frac{\partial E}{\partial b_i} = \sum_k \frac{\partial E}{\partial y_k} \frac{\partial y_k}{\partial v_k} \frac{\partial v_k}{\partial v_j} \frac{\partial y_j}{\partial v_j} \frac{\partial v_j}{\partial b_i} \tag{4.18}$$

The derivatives are the same, with the addition of the last, regarding the bias:

$$\frac{\partial v_j}{\partial b_i} = 1 \tag{4.19}$$

Following this formula and extensions of it for more hidden layers, all weights and biases can be updated according to the output of the neural network. For this update, one more formula is needed:

$$\begin{aligned} w_{t+1} &= w_t - \epsilon \frac{\partial E}{\partial w_t} \\ b_{t+1} &= b_t - \epsilon \frac{\partial E}{\partial b_t} \end{aligned} \tag{4.20}$$

t here denotes the *epoch*; the time step of the training process. η is the *learning parameter*, which can be adjusted to determine how fast the neural network is learning and adapting. A low one can make the neural network training more robust, but also slower, and there are risks of getting stuck in local minima, while a high learning parameter has the opposite traits. It is not unusual to have this learning rate be adaptive, and let it adapt according to the total error after changing the value.

This process of training is usually repeated over all available samples and multiple epochs, until either a fixed number of epochs has passed or the change in output value is sufficiently small. Another common practise is to split the available data into training, validation and testing data. The neural network is then only trained on the training data, but the validation data serves as an intermediate validation, as the neural network is also tested on this. The larger the difference in performance, the larger the danger of *overfitting* is. This phenomenon occurs when the neural network is trained too long on data, and is trying to 'memorise' this data, rather than find the general structure. The result is a perfect performance on this data, but a poor performance on very similar data. When there are signs this is occurring, the training can also be stopped. The testing data lastly is meant for a final validation to see whether the neural network indeed behaves as desired on new data. 'Performance' can be quantified with the saved error, and this formula can be slightly adapted to different error calculation methods, such as the mean sum square and the absolute sum square.

This method of back-propagation is relatively straightforward and purely depends on a first-order gradient-descent algorithm. It was the first learning method generally implemented in neural networks, and is still being used. However, more powerful and complicated algorithms exist now, a popular one being the *Levenberg-Marquadt* method, a second order method. These are considerably more difficult to implement without aid, but have significant advantages. Yet, there are dangers for instability and too aggressive weight updates.

5

WORKFLOW AND DEVELOPMENT

As mentioned before, the neural network was initially developed from scratch to get more insight in how neural networks work, and how to change them when they don't behave as expected. For this, the lectures on neural networks and self-supervised learning from the course [AE4350] Bio-inspired Artificial Intelligence and Learning for Aerospace Applications [27], the book Application of Soft Computing and Intelligent Methods in Geophysics citeHajian2018Book and the web-series Neural Networks from Scratch in Python have been of great help, though the latter only reached part 5 at the moment of working. All scripts can be found in the code appendix.

The first part is the acquisition of data, for which several options exist, according to Subsection 4.6.1. In the most straightforward manner, a Station Point class is created, and the parameters R, ρ, m, z and q are fixed at a certain value, after which the distance x is randomly chosen, the value of g is extracted theoretically and normalised. This combination of g and x is then stored. No gravity modelling takes place, and there is a perfect correspondence between the parameters and the gravity data. Note that while the parameter A is calculated, it is not used in this version.

The alternative way of data acquisition involves gravity modelling, which happens in an external program. In the case of IGMAS, the result file is a '.stations'-file, which is structured like an xml-file. There exist Python-packages, here imported as 'ET', that handle this structure efficiently and which make it possible to transform the results into an nx2 matrix, where n is the number of measurements per batch. The results are stored in a Python list, where each measurement batch is a list and an element of the total data list. This means that the total data list is a list of lists, where each element is a list of (g, x) tuples. Note that they g-values are normalised, but the x-values are by default not.

The structure of the neural network is coded in a very brief manner, using principles from object-oriented programming. Each layer is defined by the number of inputs and neurons. As only three layers are used in this version, this leads to three possible layer structures. However, as the input layer just consists of neurons with the same values as the input values, this layer is not defined separately. Each later has three main properties, those being the weights, the biases and the output. The weights are the weights of the connections between the previous layer and the current layer, the biases are property of the neurons in the current later and the output is the output value of the neurons in the current layer. Note that the neurons are not created one-by-one, but rather as a whole batch. Initialisation only happens once, with randomly generated positive weights and zero biases, after with these properties can be accessed and modified. The output is not initialised, but can be calculated after initialisation of the weights and biases and if the values of the previous layer are known. If there are n inputs and m neurons for this layer, the weight matrix is a nxm-matrix and the inputs are a $nx1$ matrix. The output of the m'th neuron is then calculated by a dot product between the input matrix and the m'th column of the weight matrix, with the m'th bias added.

The only thing the structure of the neural network now needs is the activation function, which is defined in a different class. It is chosen per layer of the neural network and has two properties; an 'output' or value and a 'diffoutput' or derivative value. Due to the simplicity of the used functions, these classes are very short. Creating these layers and generating output can then be done very easily with calls to these objects.

In the supervised variant, the output is then compared to the desired output following Equation 4.9, while the self-supervised variant uses Equation 4.11. After this, the back-propagation starts, which was not covered in the Python web-series and as a result is programmed less efficiently and more spread out. The aim is to also provide clarity in this way. The most basic example of the first example of this notebook shows the internal principles the best. This network is for demonstration purposes only, and is trained on one single sample only. The input sample is a normalised tuple without meaning, and so is the output sample. This means two input neurons, five hidden neurons and two output neurons. Logically, the first weight matrix is a 2x5-matrix, the second weight matrix is a 5x2 matrix, the input layer has as outputs the 2x1 data sample, the hidden layer consists of a 5x1 array of values and the output layer has as outputs the 2x1 final output array.

The back-propagation starts at the first layer. There is no large preference for the order of the updates, apart from one thing. From Equation 4.16 and Equation 4.17 it is known that the total gradient also depends on the weights between the hidden layer and the output layer. Thus, the input-hidden layer weights and the hidden layer biases need to be updated before the hidden-output layer weights. Thus, for both the weights and the biases, the derivatives $\frac{\partial E}{\partial y_{output}}$, $\frac{\partial y}{\partial v}$ and $\frac{\partial v}{\partial y_{hidden}}$ need to be known for every output neuron, which shows the two ways one can get to a certain connection starting at the end of the network. All these derivatives are 2x1 matrices, and the matrices are multiplied element-wise before being

added element-wise. The last two derivatives, $\frac{\partial y_{hidden}}{\partial v_{hidden}}$ and $\frac{\partial v_{hidden}}{\partial w_{input,hidden}}$ are scalars which are multiplied by the resulting scalar. This happens in batches for compactness and such that unnecessary loops are avoided. With one hidden layer, there are two layers of biases and two blocks of weights to be updated. For ease of coding, these all got their own code block. Finally, the whole process is repeated a set number of times for different epochs.

This proved to work with one sample in a supervised setting, proving that the general structure and the back-propagation was implemented correctly. Also for more samples with different values this worked out, though the performance generally decreased with a higher number of varied samples. Longer training times and more hidden neurons largely eliminated this issue, showing what one might expect. While overfitting is new danger with this action, it shows that the neural network can be tweaked if performance is lacking.

The next generation of the neural network showed many new features. One of them included the option to provide the neural network with more than one data point simultaneously. This means that the gradient for the weights and biases is calculated for a number of biases, and then averaged. In this way, one can provide the neural network with a batch of data instead of a single data point. A reason why this is done is since the switch to a self-supervised network was also done at this point. The neural network is then trying to find suitable values for q and z in Equation 4.4. The issue is that for one (g,x)-tuple, an infinite number of (q,z)-tuples are possible fits. This equation is under-constrained with only one data point, and training it on only one data point leads to a zero error, but arbitrary (q,z)-tuples. The neural network needs to use multiple data points, and having several of them available at once helps. Lastly, the change was implemented that the neural network only updates its weights and biases if this does not lead to either the z-output or the q-output being negative.

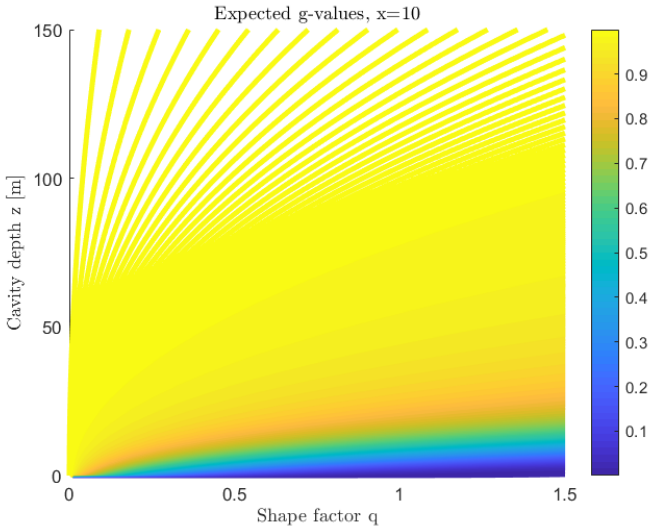
Additional new features included a subdivision of data in 'training', 'validating' and ' testing' groups. The training data is fed to the neural network during training, and based on the obtained result, the weights and biases of the neural network are changed with back-propagation. The validating data set is also used during the training, but the resulting outputs are *not* used to adapt weights and biases. The idea is that with this it can be checked whether the neural network is learning the pattern in the data sets, or merely remembering the training data set. If the loss is similar for both sets, it means that the neural network is learning the pattern well, as it responds well to new but similar data. The idea is the same with the testing data; once the neural network has finished training, it is tested on a new but similar data set, to see how it responds on this. If the response to the unknown testing data and the 'familiar' training data is close to equal, no overfitting has happened and the neural network behaves as desired in that aspect.

5.1. Intermediate results

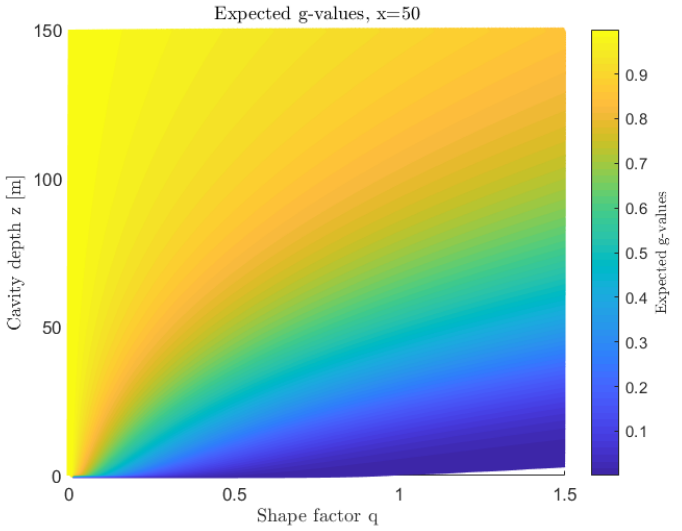
At this point, some intermediate conclusions can be drawn. With the main principles working, one can see how the performance is affected by various settings. A first conclusion is that the neural network is not able to perform well yet. After checking carefully whether the implemented formulae are as they should be, some reasons for the poor behaviour were identified, further illustrated in Figure 5.1 and Figure 5.2:

- One important reason, also why the problem is more difficult than initially expected, is still that Equation 4.4 is under-constrained with just one measurement, and that is it not straightforward to combine multiple measurements, especially with the present assumptions and noise. It is possible that the simple structure of the present neural network is not able to cope with this in the right way.
- Another issue is the complicated behaviour of the derivatives. While they are smooth and well-defined, they also pose problems. The derivatives of Equation 4.4 have extreme values for very low q and z, and the function values of Equation 4.4 show large gradients there. However, outside of this region, the derivatives are rather low, certainly in the region $0.5 \leq q \leq 1.5$ and $20 \leq z \leq 100$, the region of interest mostly. This means updating will happen slowly, and even slower when the correct value is approached. While this is a good sign for conversion, it also makes the learning process very slow.
- Due to the nature of these functions and their derivatives, errors quickly reduce to low values (in the order of 10^{-4} with original values in the order of 10^{0}), while the output parameters q and z are still in the range of 10 to 50% off. This behaviour leads to a misleading effect, implying a better result than actually is the case. Thus, the neural network easily gets stuck in a configuration which leads to very low errors for some inputs (but are still far off from the desired values) and quite low values for other inputs, which is a configuration still far from the global optimum.
- Additionally, note that all derivatives to z all have positive function evaluations, while all derivatives to q have negative function evaluations. In order to have some control over the direction in which weights and biases are updated, the error needs to be able to be negative or positive. Unfortunately, due to the problem of the equation being under-constrained, the output values can for example be both too high, but the calculated g can still be lower than the actual g.

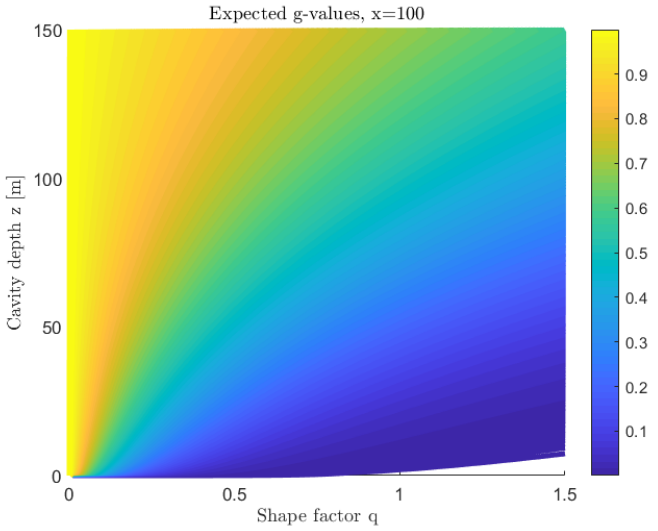
- Lastly, there is the issue that with multiple derivatives and input, it often happens that updates on biases and weights are counter-acting each other, making learning even slower.



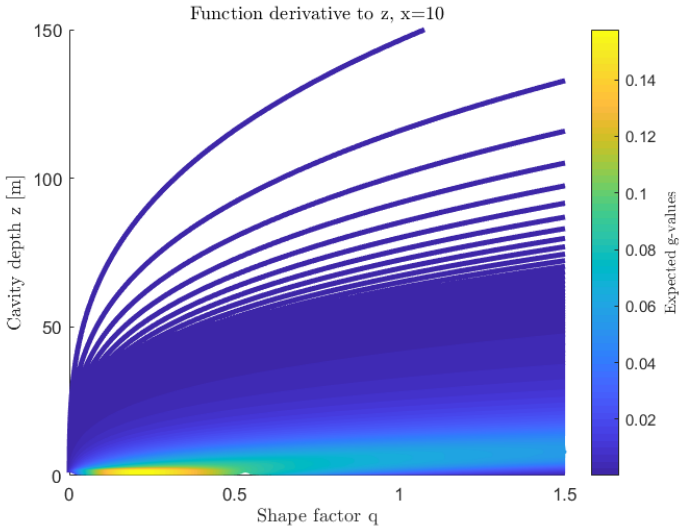
(a) Gravity signal for certain parameter combinations



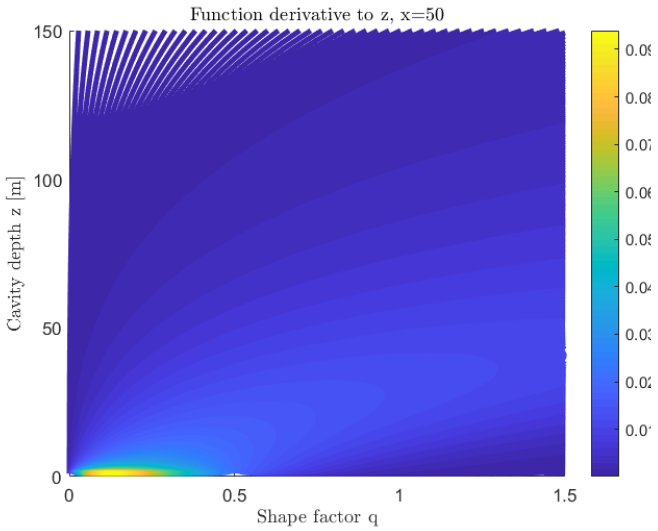
(b) Gravity signal for certain parameter combinations



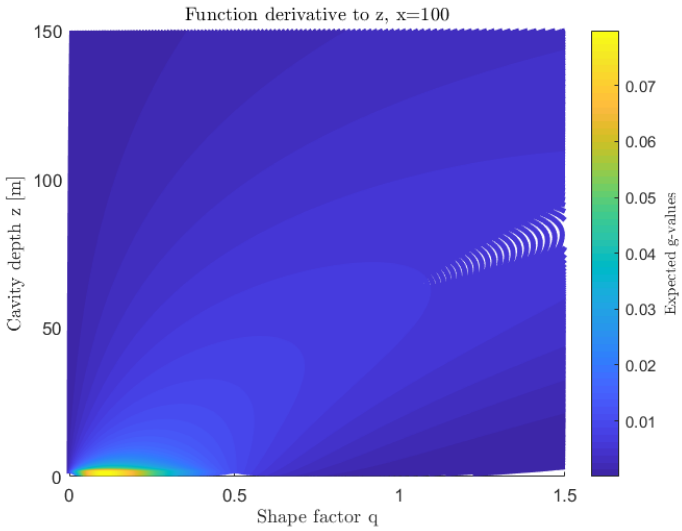
(c) Gravity signal for certain parameter combinations



(d) Gravity signal, derivative to z for certain parameter combinations

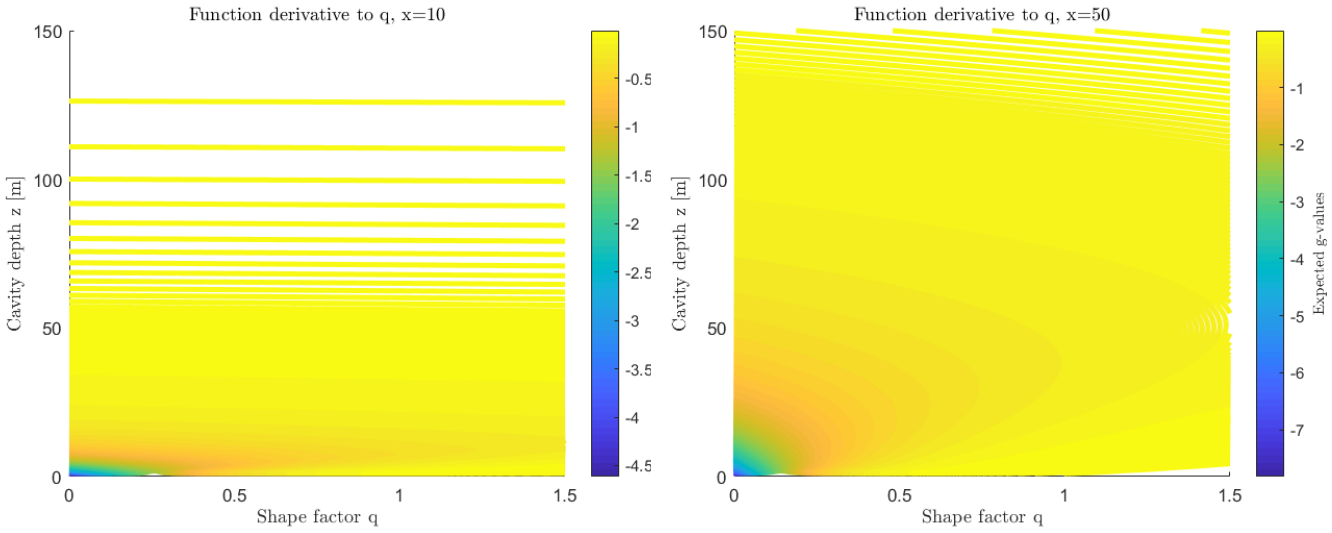


(e) Gravity signal, derivative to z for certain parameter combinations



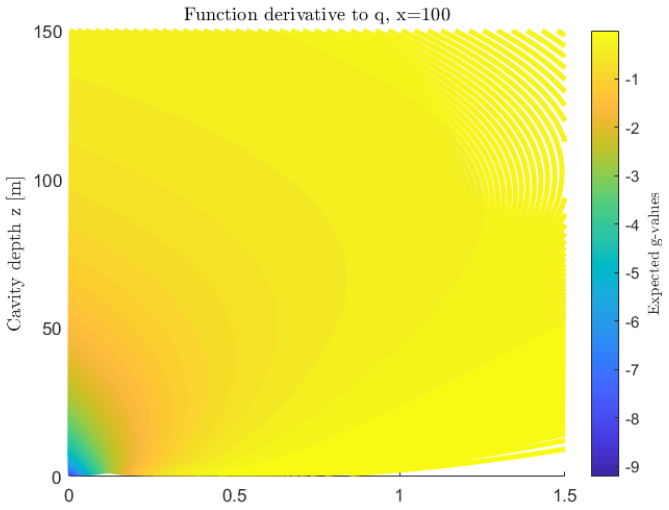
(f) Gravity signal, derivative to z for certain parameter combinations

Figure 5.1: Function values and gradients for certain parameter combinations



(a) Gravity signal, derivative to q for certain parameter combinations

(b) Gravity signal, derivative to q for certain parameter combinations



(c) Gravity signal, derivative to q for certain parameter combinations

Figure 5.2: Gradients for certain parameter combinations

Due to the issues encountered with this simple steepest descent approach, it seems more complicated and advanced algorithms are needed to make the neural network behave as desired. With current implementation, on few samples the error is consistently in the range of 10% to 50%. This means the algorithms do the right thing, but not good enough. Before switching to neural network packages with more functionalities, there is another test that can be performed, consisting of a structure with only one output neuron indicating the cavity depth. As there are only three 'valid' options for the shape factor q, it is possible to make three separate self-supervised neural networks, each assuming a specific value for the shape factor q. Given a batch of data of the same cavity, the neural networks should all give the same or a very similar output for all values in the same batch. However, for only one network the fit of the weights and biases will fit to the input. If the input for a spherical cavity is provided to a neural network assuming a vertical cylindrical cavity, it is not possible to adjust the neural network such that a comparable error is obtained for all inputs. Thus, looking at the spread of the final results, one can say something about the expected cavity shape. Of course, this method works worse for shapes which are not perfect spheres or cylinders.

The next step is then to create three separate neural networks with one output neuron, one for each value of q and thus one for each shape, and test them on data of different shapes. These neural networks have the same functionalities as aforementioned networks. Given this approach, the goal of the neural network is to solve the following equation:

$$z = \sqrt{\frac{g_n^{1/q} \cdot x^2}{(1 - g_n^{1/q})}} \tag{5.1}$$

Which is Equation 4.4 solved for z. The problem is not under-constrained anymore and is quite simple in its form; one could decide not to use a neural network for this, but rather a simple solving algorithm. The power of this neural network

method however is that it can combine data of the same batch, which does not necessarily give the same z-value following this formula (for example, due to noise or approximations) and that the spread in the values says something about how reliable the assumed q-value is. however, this can also be performed with other, more robust methods, meaning that this implementation is more for learning and demonstration purposes.

For the two-dimensional neural network, it is thus advised to use an existing framework and adapt the loss-function to Equation 4.4. Two frameworks were used for this attempt, namely the Neural Network Tool of MatLab in its Deep Learning Toolbox, and the Tensorflow package with Keras in Python. Despite many attempts with both packages, only errors and nonsensical values came out of these attempts. Other alternative loss-functions were for example possible to implement in Tensorflow, but with Equation 4.4 and its dependency on input and output in various ways, adapting this to the tensor data frames of Tensorflow became very complicated. Limited knowledge of these packages, limited available time and the availability of the self-developed program resulted in the choice to continue with the development of aforementioned program, despite its simplicity and limitations. Most importantly, quantitative analysis can still be performed, which is mostly discussed in Chapter 6.

6

RESULTS

In this section, the results of testing and improving the neural network are shown. Included are analyses of the influence of the number of hidden neurons, the number of epochs, the starting learning rate and the data set size on the performance of the neural network. This performance is both measured in terms of the final loss according to the error Equation 4.11 as well in other metrics as the spread of results. Apart from this, it is analysed how much noise can be put on top of the data with the neural network still being able to process the data. Furthermore, other comparisons with similar architectures are made, and finally the performance of the neural network with the best combination of settings is tested on increasingly more realistic data for the situation of buried lava tubes on the Moon.

6.1. STARTING POINT

The code for this part of the project can be found here under the name 'ANN for Gravity Signals - SSL'. The code at this point is based on a self-supervised learning principle with an adapted loss function, one layer is implemented with a number of neurons that can be changed. The learning rate can be set initially, but also changes when a change is made to the structure of the neural network which does not improve results. Thus, there is also some check present which looks at this. Updating of weights and biases happens through back-propagation. Due to the way these features have been implemented, both the number of layers and the learning algorithm are considered fixed and not changeable in the time frame of this project. Parameters which are changeable are as follows:

- The number of hidden neurons
- The initial learning rate
- The number of epochs
- The size of the data set

The activation function is also changeable per layer, but the researches outlined in the theory have mostly recommended the ReLu and linear activation function for the hidden and the output layer respectively, and investigating all possible combinations of common activation functions was not considered worth the time.

While a complete analysis of these options would include a set of options per parameter and then a training for all possible combinations of options, this is also very time-consuming and creates a lot of data, much of it being not useful. Experience during earlier projects, mostly for the course AE4889 Special Topics in Astrodynamics from TU Delft, have shown that a reasonable start is to train the network for all options of a parameter individually, and assume a 'default' combination of parameters for comparison. The possible influences of one parameter on others are lost this way, but the upside is that a wider scale of options per parameter can be investigated. The used default parameters for this analysis are shown in table 6.1.

Table 6.1: Default values parameters for the neural network setting analysis

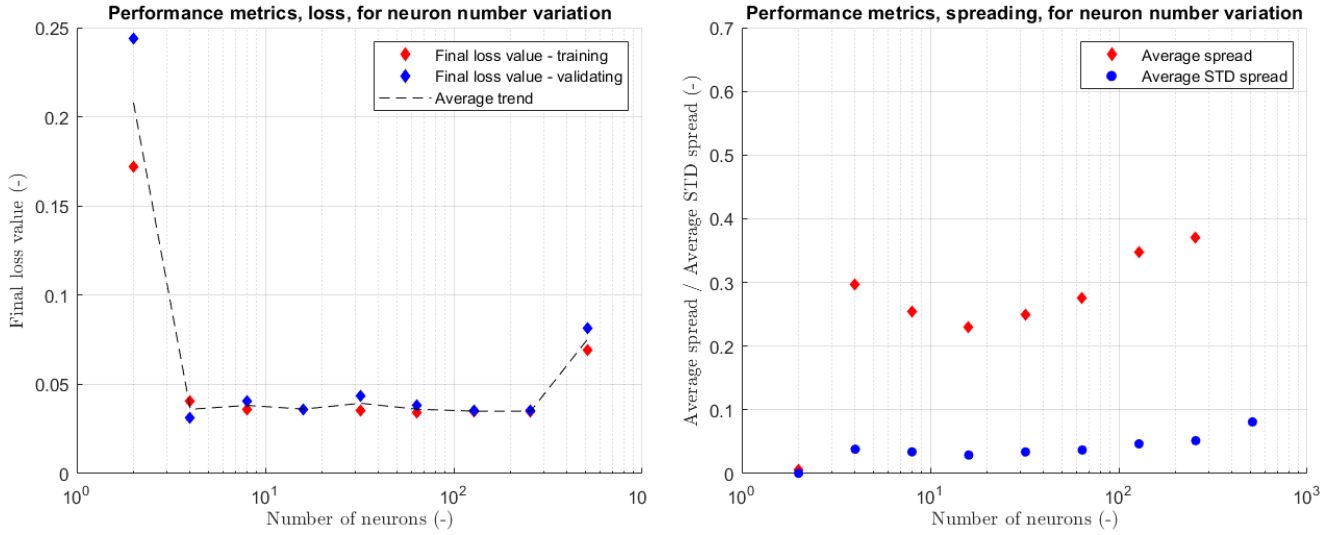
Hidden neuron number	64
Number of epochs	20
Number of data samples	100
Number of data points per sample	15
Initial learning rate	0.25
Neural network architectures	(2,64,2)
Activation functions	ReLU, Linear

6.2. NEURON NUMBER VARIATION

An important factor about neural networks which has not been discussed yet is the time needed to train the neural network and do an analysis. For large architectures and complex tasks, this time can be up to several months. Yet, during an analysis of settings, many runs have to be completed. Timing is usually an issue, and so has it been during this project. Thus, it pays of to first analyse settings which influence the training time, to see if a default setting can be changed to a value leading to lower training times without a (large) compromise in performance. In general, these are the number of hidden neurons, the size of the data set and the number of epochs. This section concerns the analysis of the influence of the number of hidden neurons on the performance of the neural network.

As metrics for this analysis, first of all the loss and validation loss at the final epoch is taken. Recall then the data structure; 100 sets of 15 data tuples of the same configuration of z and q (also ρ and R are set, but they are not relevant). This means that all 15 data tuples (x,g) of one set are different, but have the same (z,q) tuple associated with it. If the neural network understands this, all 15 data points will give the same output when presented to the neural network. Often, this is not the case, and some kind of 'spread' is present. This spread is quantified by calculating the extent of the spread for each data set as well as calculating the standard deviation. These two values are two more metrics. Lastly, it is important to always have a visual check of the situation, as will become clear.

As for the number of neurons, the values 2, 4, 8, 16, 32, 64, 128, 256 and 512 were tested, with the notion that the last option required a training time of more than two hours, making it not viable for practical use during this analysis. The main results are summarised in Figure 6.1. There are a few interesting trends visible here. First of all, the loss is largely constant over the number of neurons, except for the first and last entry. This shows that for this loss function, the number of neurons does not seem to matter very much for the final result. In Figure 6.1b something else is visible; some kind of trend regarding the spread. A lower spread combined with a good general trend is fortunate, showing that the number of neurons does have some influence on how good the final results are. The final entry - with 512 neurons - gives surprisingly poor results. The higher loss can be attributed to the fact that it is inherently random and dependent on the random initialisation, but this does not reflect the full picture. First of all, comparing for example Figure 6.2c and Figure 6.2d, the results for 64 and 512 hidden neurons respectively, we see that the structure of these graphs is very similar. While these neural networks are different, they have been trained on the same data set, which is visible in the resulting graphs. This shows that the random effect is not significantly present and that the anomaly of 512 neurons in Figure 6.1a is due to something else. What could be the case is that the number of epochs and data set size are not high enough for the high number of neurons to be trained properly. yet, the loss value of the neural network with 512 hidden neurons is fairly constant from about the fifth epoch.



(a) Trend of the final loss on the training and validating set with settings from Table 6.1, apart from the neuron number

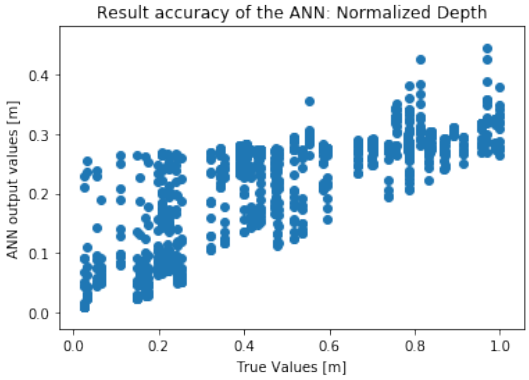
(b) Trend of the spread and the standard deviation of the spread on the training and validating set with settings from Table 6.1, apart from the neuron number

Figure 6.1: Performance metric for neuron number variations

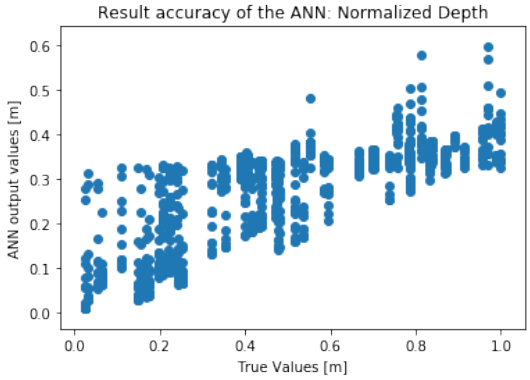
Also the neural network with only 2 hidden neurons gives intriguing results. Figure 6.1b seems to show excellent results for this network, but a look at Figure 6.3 shows differently. This figure shows the result of using the training data set as the input for the already trained neural network. The results are far off from what they should be, and it is clear that this simple neural network architecture of (2,2,2) is not fit for this problem.

Looking back at Figure 6.2, a few remarks must be made. The graphs show the result of using training data as input for the

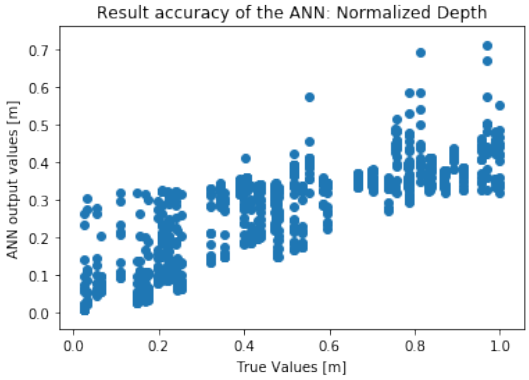
already trained neural networks. It is intriguing that they all show the exact same structure of results, but are not exactly equal (outliers are different for the different configurations). These networks are all trained on the same data set, but they start with a random initialisation and data points for training are chosen in random order. This indicates that during training, random effects play a very minor role. However, the shape of the resulting graphs is also very peculiar, with two clear straight lines visible serving as bottom and top boundary for the spread. Note also that for the neural network with 512 hidden neurons, the top boundary starts at 0.6, significantly higher than the other result graphs. Ideally, one would see a perfectly straight line with a slope of 1, and while the straight line is somewhat visible, the slope is much too low. However, once this trend has been identified, this can be corrected for. Even with this correction, the depth estimation is only good for high depths.



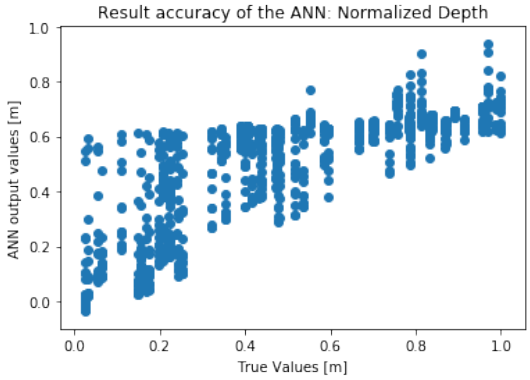
(a) Output of the ANN (64 hidden neurons) when the training data is used as input with settings from Table 6.1, apart from the neuron number



(b) Output of the ANN (128 hidden neurons) when the training data is used as input with settings from Table 6.1, apart from the neuron number



(c) Output of the ANN (256 hidden neurons) when the training data is used as input with settings from Table 6.1, apart from the neuron number



(d) Output of the ANN (512 hidden neurons) when the training data is used as input with settings from Table 6.1, apart from the neuron number

Figure 6.2: Final output of various ANN's with training data as input

The only reason the results with the training set as input have been used rather than the one with the testing set as input is that these give more data points and a clearer visualisation. In Figure 6.4a and Figure 6.4b the difference between the training data and testing data results can be seen. The structure of the results is very similar and nearly identical, but the testing data results are more sparse, as fewer data points have been used here. This indicates that over-fitting is not an issue here; even though the neural network is not familiar with the testing data, the results are very similar to the results from the data where the neural network has been trained on.

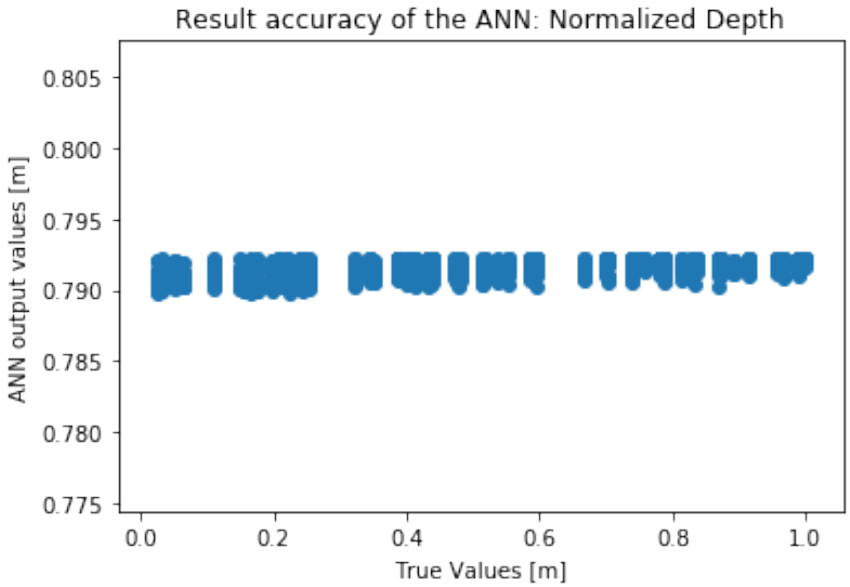
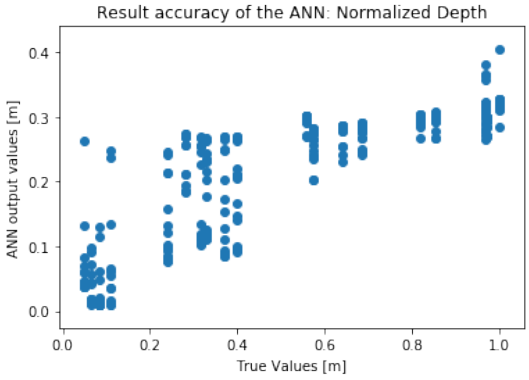
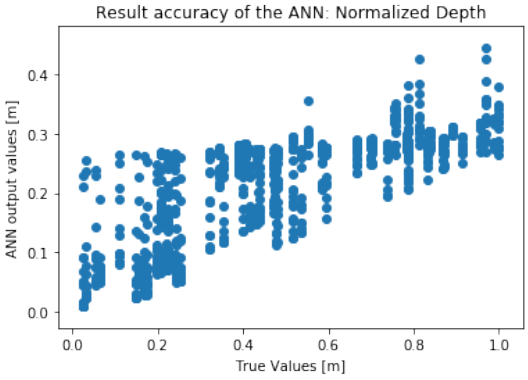


Figure 6.3: Output of the ANN (2 hidden neurons) when the training data is used as input with settings from Table 6.1, apart from the neuron number



(a) Output of the ANN (64 hidden neurons) when the testing data is used as input with settings from Table 6.1, apart from the neuron number



(b) Output of the ANN (64 hidden neurons) when the training data is used as input with settings from Table 6.1, apart from the neuron number

Figure 6.4: Performance metric for neuron number variations, specifically the difference between training and testing data

So far, only the results of the depth estimation have been shown, which is only one of the output values of the artificial neural network. This is since the results for the shape estimation have unfortunately been disappointing. An example can be seen in Figure 6.5, which is the result of the neural network with 512 hidden neurons. The order of magnitude is correct and the spread of the output shape factors is close to what it should be, but there is almost no difference between the different actual options. A perfect result would show only three dots; at (0.5, 0.5), (1, 1) and (3/2, 3/2). It is clear that with default settings, the shape factor q is impossible to reliably predict with this neural network. After this settings analysis, a closer look at this is given in Section 6.7.

Finally, there is one other interesting observation to make, shown in Figure 6.6. Here, the loss during training is shown for the network of 4 hidden neurons and for the network of 128 hidden neurons. It is visible that the loss during training for the neural network of 128 hidden neurons is smoother and decreases quicker compared to the loss of the network with 4 hidden neurons. Also, the validation and testing losses are much closer together. This effect is also visible for other low and high neuron numbers, and it shows that adding hidden neurons helps in decreasing the influence of random effects on the performance of the neural network. However, as all losses converge to a certain number at the end, the number of epochs also assured minimal influence from random effects. Such a random effect is the initialisation of the weight matrices, which can give both negative and positive values. Starting with negative values can lead to neurons not transmitting information due to the choice of the ReLU activation function, halting the ANN training.

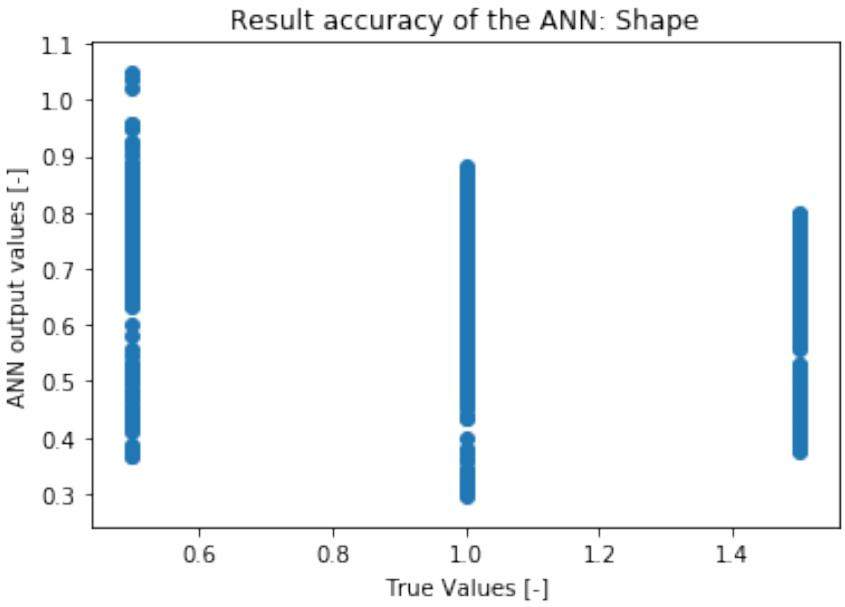
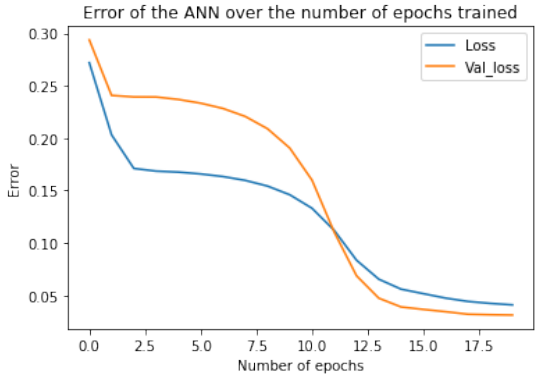
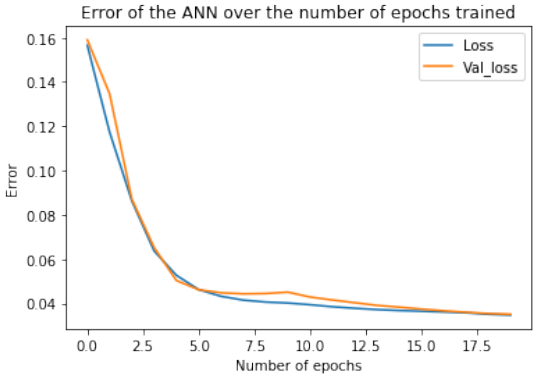


Figure 6.5: Output of the ANN (512 hidden neurons) when the training data is used as input with settings from Table 6.1, apart from the neuron number



(a) Training and validating loss of the ANN (4 hidden neurons) during training with settings from Table 6.1, apart from the neuron number



(b) Training and validating loss of the ANN (128 hidden neurons) during training with settings from Table 6.1, apart from the neuron number

Figure 6.6: Loss during training for the network of 4 hidden neurons and the one of 512 hidden neurons

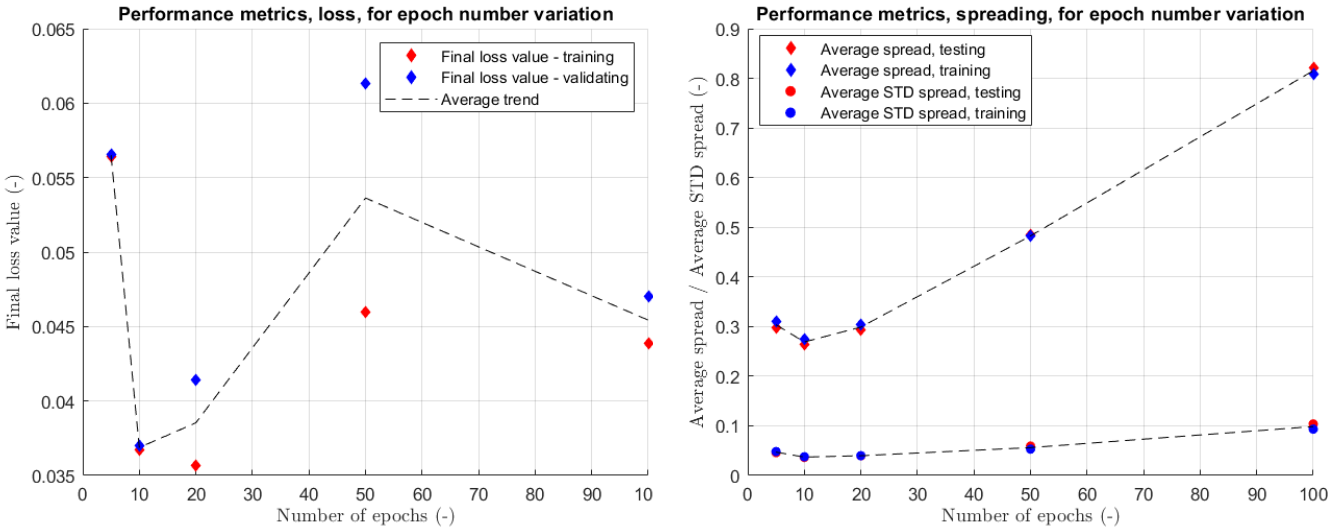
In the end, mostly because of the results of Figure 6.1a and Figure 6.1b as well as visual inspection of the graphs, it was chosen that the hidden neuron number of 64 is a good choice for further use. The average spread as well as the average standard deviation is low, and the final loss is slightly lower than for other neuron numbers. However, the results are quite close for neuron numbers 16, 32 and 64. The idea is that with a slightly more complex network (e.g. higher neuron numbers), quirks in terms of results have a higher chance of being removed later on.

6.3. Epoch number variation

After the analysis of the neuron number, the epoch number is a good parameter to investigate. The same data set can be used still for good comparison, and the epoch number has a linear relation with the training time. Thus, one would like a low epoch number. Additionally, it is often seen that from a certain epoch number, almost no improvement in terms of loss can be seen; this is also visible in Figure 6.6a and Figure 6.6b. From this point on, the risk of overfitting exists, where the neural network gets better performance on the training data set, but not on the validating or testing data set. Thus, it is of vital importance to choose a fitting epoch number.

First of all, the same performance metrics as earlier were used, and the result of using these performance metrics is shown in Figure 6.7. There are a few remarkable results visible requiring discussion. First of all, the graph of Figure 6.7a shows that there is no clear trend is visible for the final loss value over the epoch number. Contrary to what one may expect, the graph does not strictly decrease, meaning that the performance does not increase with more training. Random effects begin to have an important role here, but in an unexpected way, which is discussed a bit further in this section. Note also that in Figure 6.7a, the validating final loss value is always higher than the training loss final value, which is not the case

in Figure 6.1a. This is also an indication of overfitting, as the neural network always performs better on the data set where it trains on. Figure 6.7b on the other hand shows very clear trends for both the spread and its standard deviation. In this case, the values for the training data set and the testing data set are very close together, showing that there is still a general trend happening. In both graphs, the epoch number of 10 (rather than the default number of 20) gives the best results, while also cutting the training time in half.

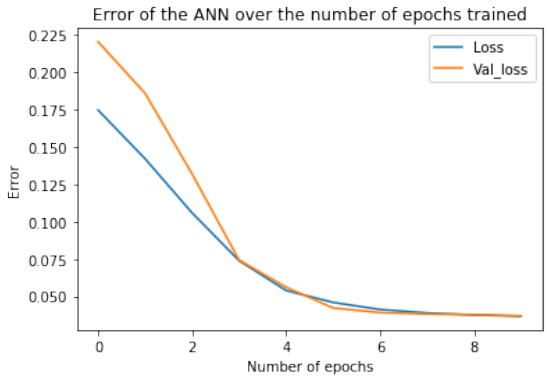


(a) Trend of the final loss on the training and validating set with settings from Table 6.1, apart from the epoch number

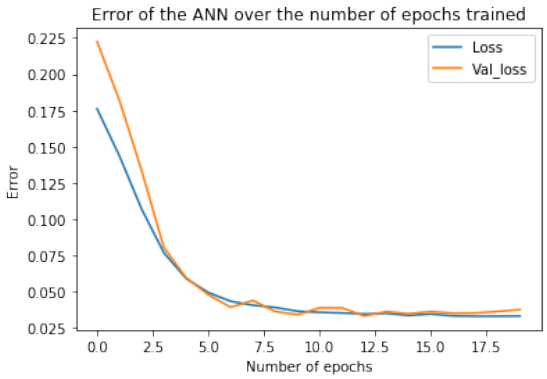
(b) Trend of the spread and the standard deviation of the spread on the training and validating set with settings from Table 6.1, apart from the epoch number

Figure 6.7: Performance metric for epoch number variations

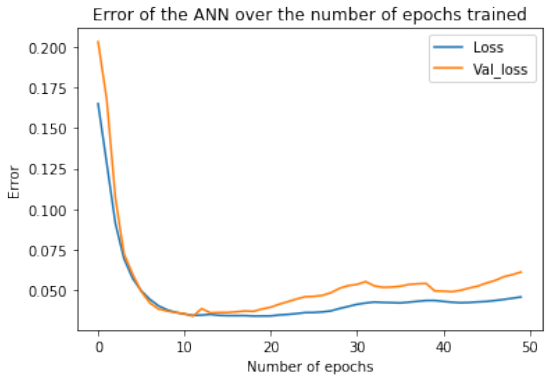
It is also interesting to look at the loss graphs in Figure 6.8. Note that these graphs are all the result of different runs; while the one of 100 epochs could also have been stopped at 10, 20 and 50 epochs for these results, there was a preference for using separate runs to look at possible differences, which should be the result of random effects. In all graphs the validation loss is higher than the training loss, and while it catches up at about epoch number 5-10, the lines diverge again at usually epoch number 20. In general, the loss goes up quite consistently at epoch number 20, and the loss becomes quite jittery after this point, while generally staying at about the same level. Regarding loss alone, an epoch number of 10 to 20 is advisable, though there is more to say about this.



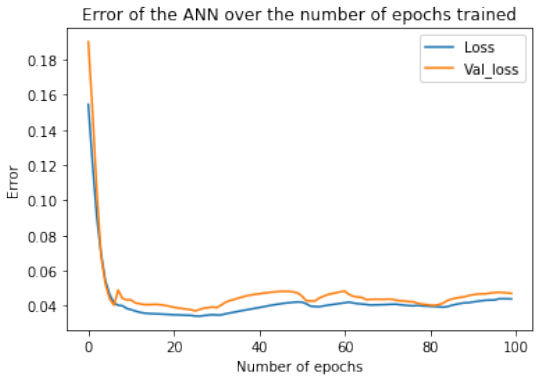
(a) Training and validating loss of the ANN (10 training epochs) during training with settings from Table 6.1, apart from the epoch number



(b) Training and validating loss of the ANN (20 training epochs) during training with settings from Table 6.1, apart from the epoch number



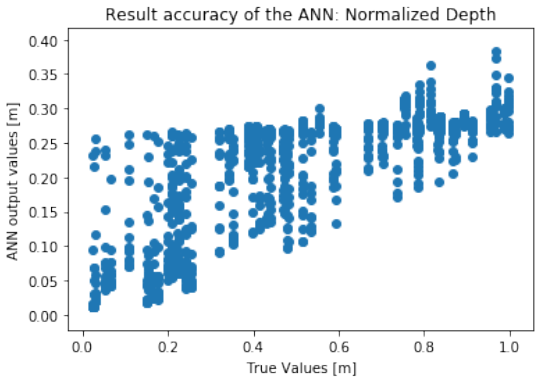
(c) Training and validating loss of the ANN (50 training epochs) during training with settings from Table 6.1, apart from the epoch number



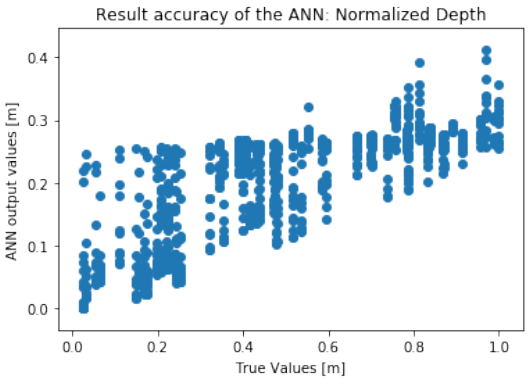
(d) Training and validating loss of the ANN (100 training epochs) during training with settings from Table 6.1, apart from the epoch number

Figure 6.8: Loss during training for 10, 20, 50 and 100 epochs

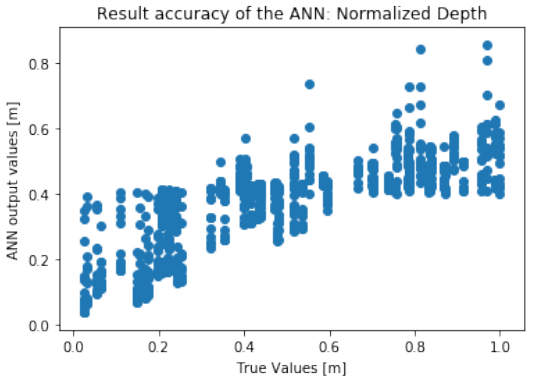
Looking at the results in Figure 6.9 of the output of the trained neural networks at various epoch numbers. As with the neuron number variation, the results have very similar appearances and also look very much like the results from that section, as the same data set is used. Thus, there is seemingly little reason for the loss to change after about 20 epochs. However, looking closely it can be seen in Figure 6.9c and Figure 6.9d that the horizontal 'plateau' is much higher, at about 0.4 and 0.6 respectively compared to the 0.3 of previous results. In general, the graph appears stretched compared to the graphs in Figure 6.9a and Figure 6.9b. In Figure 6.9d a linear line with slope 1 can even somewhat be discerned, indicating a good regression. Yet, the final loss value for this situation is higher than for epoch numbers 10 and 20. This is likely a combination of a higher spread for the situation with 100 epochs (as is made clear in Figure 6.7b) and the low derivatives in Figure 5.1e and Figure 5.1e, indicating that the neural network can see little difference in terms of loss between the situations of Figure 6.9a and Figure 6.9a. Yet, it is not completely clear why the spread occurs in the first place, and also not why the output values become higher with more epochs. As the test and train results are also extremely similar (see Figure 6.7b), overfitting does not quite explain this phenomenon. It seems that the loss can not decrease due to the inherent spread present in the results. Tests were performed with fewer samples per data set (for example, training with 100x5 data points), but this did not have an effect on the standard deviation on the spread. Also, this spread is not due to training in a fixed sequence, as the data set to train on (with the 15 samples) is chosen randomly every time, and the initialisation of the neural network is random each time as well.



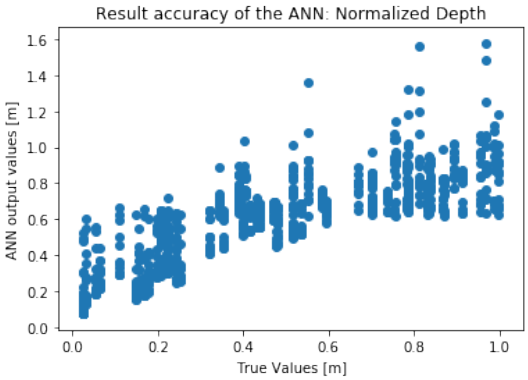
(a) Output of the ANN (10 training epochs) when the training data is used as input with settings from Table 6.1, apart from the neuron number



(b) Output of the ANN (20 training epochs) when the training data is used as input with settings from Table 6.1, apart from the neuron number



(c) Output of the ANN (50 training epochs) when the training data is used as input with settings from Table 6.1, apart from the neuron number



(d) Output of the ANN (100 training epochs) when the training data is used as input with settings from Table 6.1, apart from the neuron number

Figure 6.9: Final output of various ANN's with training data as input

As with the neuron number testing, the results regarding the shape factor q were not useful again, just showing three vertical bars. Evidently, the spread present here is too large and without structure to be useful. It seems that obtaining the shape factor with the loss function for self-supervised learning is much more difficult than obtaining the depth value.

Mostly based on the results of Figure 6.7a and Figure 6.7b as well as visual inspection of other graphs, an epoch number of 10 was chosen for further usage. While particularly Figure 6.9d showed promising results with a trend more similar to a linear one with slope 1, similar result can be obtained by simply multiplying the results of Figure 6.9a with a certain factor, though one would not get the peculiar horizontal plateau in the right of the graph in Figure 6.9d, for which again little explanation can be given.

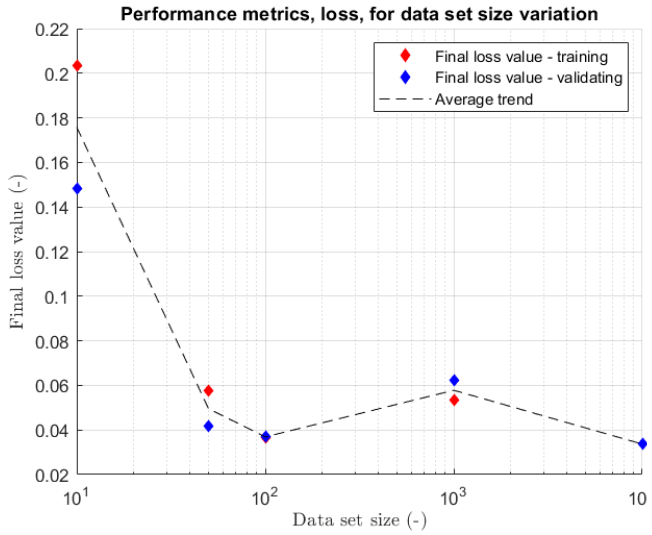
6.4. Data point variation

The last setting which significantly influences the computation time for the training of neural networks is the number of data points. Usually, it holds that more data points - if well-representative - makes the neural network perform better [4]. Yet, it can also have other effects, such as requiring more complex neural networks or more training epochs.

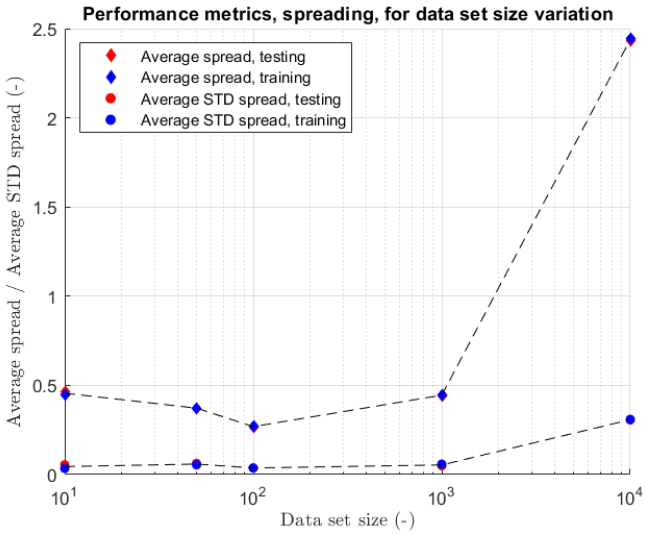
First, we look at the usual metrics for this analysis in Figure 6.10. There are clear anomalies for low and high data set sizes. The lowest data set size was 10 with 15 samples each. Evidently, this is not enough for the used neural network architecture to be properly trained. This is expected behaviour, as it is known the neural network needs a basic number of data samples to operate. The variations in loss, average spread and the standard deviation of the spread for the other data set size numbers are relatively low, except for the largest data sets of 10000x15 data points. Once again, it must be said that this configuration took more than 8 hours to train the neural network, so it is not suitable for usage right now, but the results are peculiar. While the loss reaches a relatively low number, the average spread reaches very high values, not present in any other configuration. A look at Figure 6.11 reveals why. First of all, the shape of the graph in Figure 6.11a is very peculiar; the general features of previous depth value output graphs is visible, but with a very strict and perfectly straight bottom boundary, with again a diagonal and horizontal line visible. Apparently, during training the data points are pushed towards that line from the 'bottom', but not from the 'top'. The fact that this feature shows up often is a sign that it is fundamental to the inner workings of the neural network, perhaps the loss function.

Note also that the values on both graphs in Figure 6.10a and Figure 6.10b are very high, much higher than they should be. The reason why the loss is then still so low is because of the nature of Equation 4.6; a higher value of q will decrease

the corresponding g-value. Just like earlier situations where this happened, the higher values for the depth output are accompanied by higher q-values, which makes the loss roughly constant. Yet, when the values become very high, the spread also becomes quite high, as every value is merely inflated. This leads to a very high spread and standard deviation in this spread. In general, these results are far from practical for actual use, but they give crucial insight in the internal processes of the neural network.

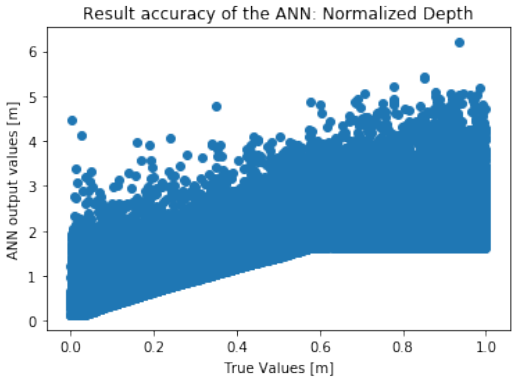


(a) Trend of the final loss on the training and validating set with settings from Table 6.1, apart from the data set size

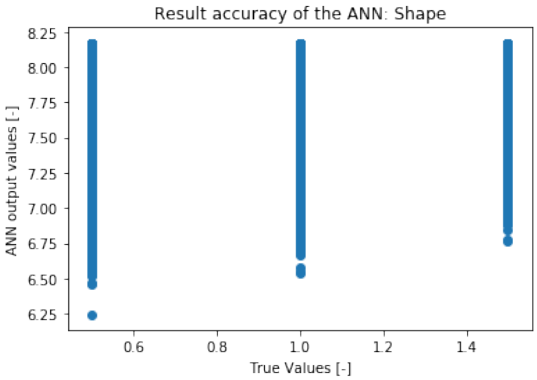


(b) Trend of the spread and the standard deviation of the spread on the training and validating set with settings from Table 6.1, apart from the data set size

Figure 6.10: Performance metric for data set size variations



(a) Depth value output of the ANN (10000x15 data points) when the training data is used as input with settings from Table 6.1, apart from the data set size



(b) Shape factor output of the ANN (10000x15 data points) when the training data is used as input with settings from Table 6.1, apart from the data set size

Figure 6.11: Resulting values from the trained neural network with 10000x15 input samples

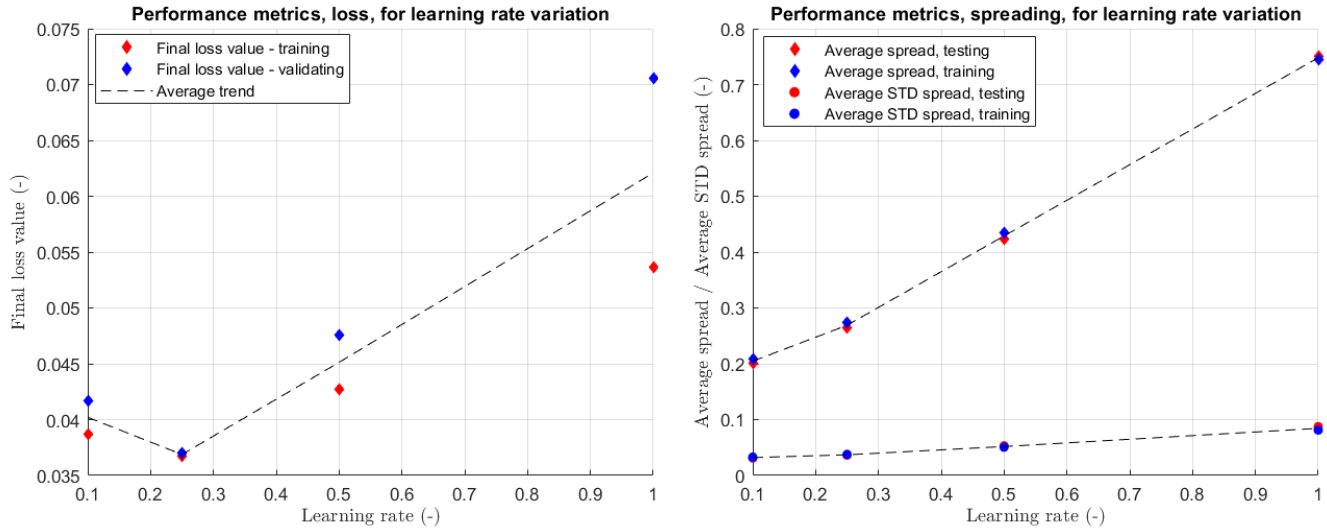
Other results from the data set size analysis are not very noteworthy and they show similar characteristics as earlier tests. From Figure 6.10a and Figure 6.10b and visual inspection of other results, it can be seen that a data set size of 100x15 is suitable for the neural network performance. As noted earlier, the '15' of the number of data point tuples per configuration has been varied as well, without insightful results. Thus, no change is applied to the default settings.

6.5. Learning rate

The learning rate can determine how fast the neural network reacts to new changes and also to what extent the neural network is prone to 'overshooting', meaning that the performance is close to convergence, but can't convergence since the neural network is not able to make small changes. This is partially accounted for by adaptive learning, meaning that the learning rate changes depending on earlier results and how good they were. Yet, since the change of weights and biases are proportional to a wide range of derivatives which can become very small, the initial learning rate can still be very important for the final result. Thus, the influence of the learning rate in this particular project is also investigated.

The main results are shown in Figure 6.12, with the usual metrics of loss, average spread and average standard deviation in spread. While there is a clear trend visible in both graphs, showing a poorer performance for higher learning rates, this

is again since both the depth output and the shape factor output of the neural network are generally higher than those of lower learning rates. While this is a step in the right direction for the depth output, again the high spread of the depth output and the poor match with the desired results regarding the shape factor result in a poor performance, according to these metrics. There are no other noteworthy results regarding this parameter, and looking at the results there is no reason to change the initial learning rate of 0.25.



(a) Trend of the final loss on the training and validating set with settings from Table 6.1, apart from the learning rate

(b) Trend of the spread and the standard deviation of the spread on the training and validating set with settings from Table 6.1, apart from the learning rate number

Figure 6.12: Performance metric for learning rate variations

6.6. Alternative loss functions

With the previous results of the parameter variation in mind, it seems that the loss function is not susceptible enough to make out the true desired trend in the results. While the straight line with constant slope is visible in most results, it is not the only clear pattern, and changing the parameters did not help finding another pattern in these results. As this occurred with every parameter variation, different random initialisations and different data sets, this seems to be due to something internal and inherent of the problem. As verification tests have been performed to ensure that the neural network itself works well (in a supervised learning setting), it must be something regarding the self-supervised learning part. The loss function is a clear candidate for possible change, yet not much change can be done as the implementation of the self-supervised learning principle depends on Equation 4.6.

Yet, there are two possible changes which can be made without losing the used principles. These are discussed in the next subsections.

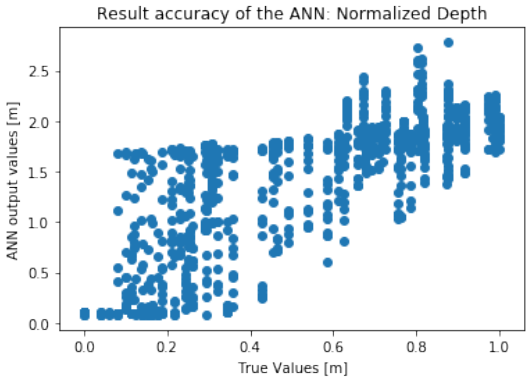
6.6.1. Using more data: a summed loss function

The current implementation is to take a data set of 15 (x,g)-tuples and then one-by-one compute a loss for them, using Equation 4.6. Thus, an output tuple (z,q) is generated, and this is used to compute for every x-value one g-value. Another approach would be to already combine all information of the data batch into the single loss, by summing the individual errors. This goes as follows:

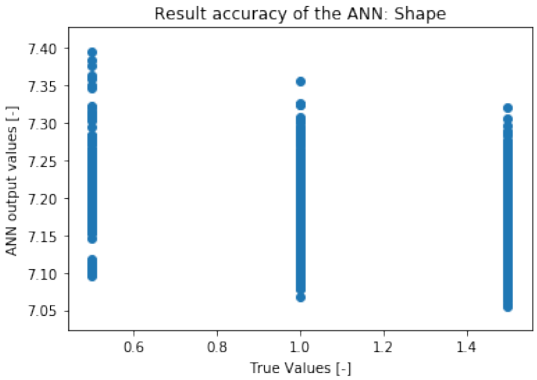
- The first (x,g)-tuple of the first data batch is used as input for the neural network
- The neural network produces an output (z,q)-tuple
- The process is repeated for the other 14 (x,g)-tuples of the first data batch
- Using the *first* (z,q)-tuple, Equation 4.6, and the *first* x-value of the data batch are used to produce a g value. The difference with the actual *first* g-value is taken for the error
- Next, the *first* (z,q)-tuple, Equation 4.6, and the *second* x-value of the data batch are used to produce a g value. The difference with the actual *second* g-value is taken and added to the error
- This process is repeated for all the 15 (x,g)-tuples of the first data batch

The end result is that the output (z,q)-tuple is tested on all the possible inputs, instead on just one. This is fortunate, as the output (z,q)-tuple of one (x,g)-tuple of the batch should match equally well with the other (x,g)-tuples of the same data batch of 15.

In practise, this means the inclusion of an extra loop for summing the variables. Note that the results and the intended changes for back-propagation are already averages and combined, so it is expected that there is not a large difference with earlier results. Results are generated with a neuron network with default settings and 10 epochs and are shown in Figure 6.13 and it is visible that there is some difference, namely in the values being too high. Yet, the shape of the figures is again very similar to earlier results, the training and testing data give similar results and the loss over the epoch number gives a clear convergence. Evidently, this alternative loss function does not give better results.



(a) Final depth output values with settings from Table 6.1, apart from the loss function



(b) Final shape factor output with settings from Table 6.1, apart from the loss function

Figure 6.13: Final output values for the alternative summing loss function variations

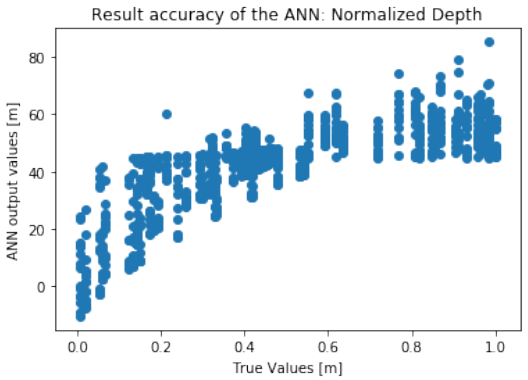
6.6.2. Solving for x instead of g

Both Equation 4.4 and Equation 4.6 are a result of solving equations to the gravitational acceleration g. Yet, these formulae, and specifically Equation 4.6, can also be solved for x. This gives the following formula:

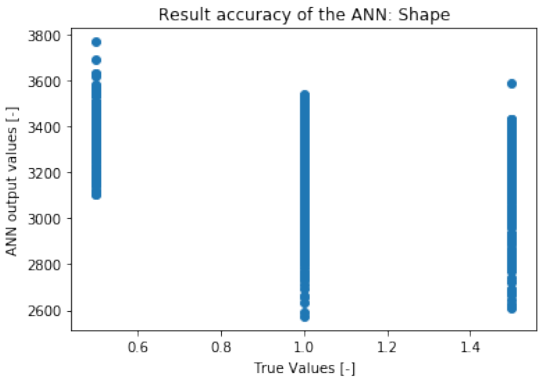
$$x = z \cdot \sqrt{g_n^{-1/q} - 1} \tag{6.1}$$

Analogously to the original loss formula, an (z,q)-tuple outputted by the neural network can be used to estimate the original x-value and compare it to the actual x-value. A caveat to be made here is that this function has different derivatives to z and q, which are relevant to the back-propagation. In the case of this formula, derivatives are very steep for low q-values and have low values for other values.

With a neural network with default values, 10 epochs for learning and this alternative loss function, new results were generated and shown in Figure 6.14. Similar structures as before can be seen, though Figure 6.14a has a rather thing arc present, rather than a thick straight line with a bend halfway. Thus, the spread is relatively low. Yet, the values for both the depth and the shape factor are extremely high, showing that this loss function is also not suitable for general usage. It was also noticed that this method with alternative loss function is susceptible to having extremely high or even negative losses for some data sets, which makes it less reliable.



(a) Final depth output values with settings from Table 6.1, apart from the loss function



(b) Final shape factor output with settings from Table 6.1, apart from the loss function

Figure 6.14: Final output values for the alternative summing loss function variations

6.7. Alternative 1D neural network

Until this point, the ANN output for the depth value have been peculiar, though somewhat useful. This is not the case for the ANN output for the shape factor, this output has been surprisingly useless. The problem seems to lie in the dimension of the loss function; Equation 4.6 has two unknowns, q and z, which the neural network attempts to find. For a given x and g, an infinite number of (z,q)-tuples are possible. To fix this issue, data was given in batches of 15 and the weight and bias updates are results of combinations of updates from all these input tuples Yet, this did not resolve the issue. However, in this unique case there is another solution. One of the parameters of the problem, namely q, has a very limited range; only the values 1/2, 1 and 3/2 are allowed as shape factors. This brings us to the option of making three separate neural networks, one for each shape, and train them on their respective cavity shapes. The idea is then, that testing a neural network trained for spherical cavities on data from spherical cavities gives good performance in terms of the used metrics, and poor results on data from other shapes of cavities.

The adaptations that need to be made for this to work are minor. The number of output neurons decreases from two to one, there is one less derivative needed, and in the loss function the value of q is assumed. When the data set is generated, also one value of q is assumed. Three neural networks are formed this way, one for each shape, and they are trained on their respective shape data. Then, they are tested on data from all three shapes. The results are shown in Figure 6.15 and they show the first case where the shape factor can be obtained from the results. The orange line is a good calibration, as the line shows a 'perfect response', where the depth value as output of the ANN is exactly the desired output. As can be seen, when the shape factor of the training and testing data set match, the orange line matches the bottom of the distribution for the first part of the graph almost perfectly. Interestingly, the distributions have the same shape as before, despite this being an one-dimensional problem now. Again, there is a clear point visible where the slope of the distribution of results changes abruptly, a point we call 'inflection point' from now on. Apparently, when training a sphere ANN on sphere data and testing it on sphere data, an inflection point of 0.6 is obtained. For the horizontal cylinder and corresponding ANN, this is at about 0.55 and for the vertical cylinder and corresponding ANN this is at about 0.45. These values are close together, but it is seen that when an ANN is trained on non-corresponding data, the inflection point is very different than what it is supposed to be. Also, the orange line matches the distribution much poorer. Rather than spread and loss, in this case the position of the inflection point and the fit to a line with slope 1 are better metrics, though these metrics require manual visual inspection.

While these results seem very promising, a note of caution must be made. While the results are discernible, the data is still of perfect spheres and cylinders, without any anomalies or perturbations. A noise analysis is described in Section 6.8 to see how well the method described above works. A more important restriction however, is that one needs the full graph for conclusions. The approach in a real life situation is to obtain data, use this as input for the three ANN's, and then do a visual comparison with Figure 6.15, or preferably one with larger data sets or added noise. If the obtained three graphs match one horizontal row, the shape factor can be obtained. The major limitation is that one needs a representative data set, implying measurements of hundreds of cavities with equal shapes. This is far from what happens in a field survey; here one collects many measurements of the same cavity. Using this data as ANN input merely gives a vertical bar with spread. It is still possible to do this with the three ANN's and then do a visual comparison to see which shape this fits the best, but this method is far from fail-proof. An instance in which this works is, looking at Figure 6.15, when data from a spherical cavity with normalised depth is obtained. Output values range in the interval [0.6,0.8] for the sphere ANN, in the interval [0.3,0.55] for the H cylinder ANN and in the interval [0.2,0.5] for the V cylinder ANN. As the actual depth and shape are not known, these values don't say much, but fitting them in the graphs of Figure 6.15 is only possible on the top row, indicating a spherical cavity. If many data samples have been taken from this cavity, the depth can even be obtained fairly accurately due to the sharp boundaries of the distribution. Thus, in this instance data from a single configuration could be coupled to the correct shape and depth with neural networks based on self-supervised learning, but there are many more instances where this is not the case. Yet, these results from the 1D neural networks are important and promising on their own.

6.8. Noise analysis

With the main settings of the ANN analysed, and finding options to obtain both the depth value and the shape factor from a data set using the self-supervised ANN, albeit with limited accuracy, it is time to use different data. So far, the data was obtained using Equation 4.4 itself, by randomly generating combinations. Yet, this data is not representative of real-life gravity survey data.

While gravity modelling programs can be used to generate the next sets of data batches, it is better to think critically about the data tuples used as input. These are tuples of a distance value and a gravity acceleration. While the location of this measurement point is important, a gravity modelling program generates much more data than needed, producing contour maps and pictures. One could decide to model the elementary shapes described before in a gravity modelling program and either make them courser or add random density anomalies in the survey region. Alternatively, one can use the previously generated 'perfect-shape' data sets, and add artificial noise on these data points. This is not a structural

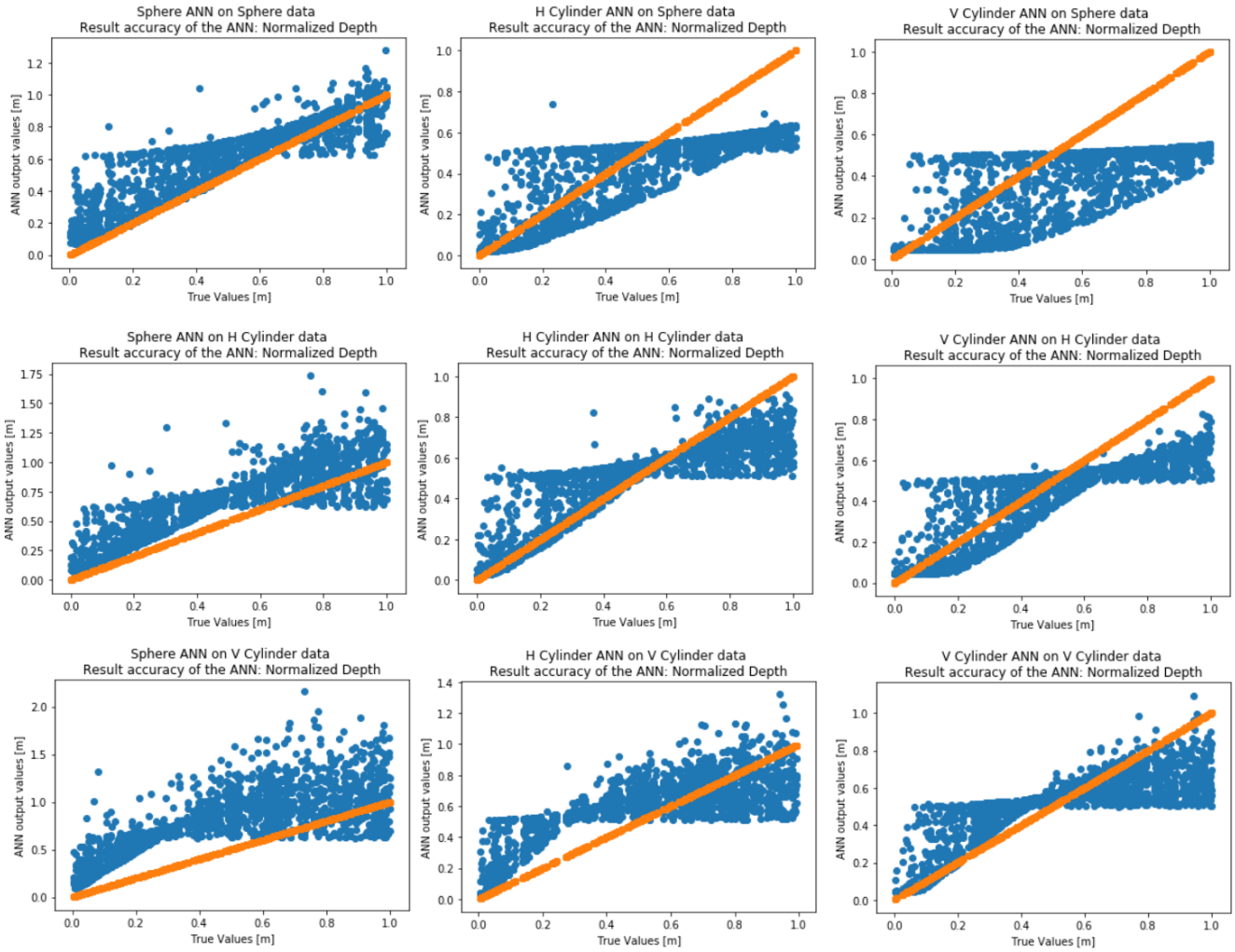


Figure 6.15: Results of usage of various data sets on various ANN's with default settings, each assuming a different shape factor. The orange line shows a perfect response.

approach, as this does not quite represent actual shapes, but on the other hand, the symmetry of the problem is used as an advantage. This, in the sense that it is not an issue that two similar data points could have noise with an opposite sign added to them, as these data points could be on opposite sides of the maximum density anomaly. Adding noise on results from a gravity modelling software is different; here you have data points related to certain locations, and some errors are correlated with location. Thus, this naive yet computationally efficient method takes measurement noise, density anomalies, imperfect shapes, and miscellaneous noise contribution into account by not having connection with location. Yet, it is not clear what a typical noise level for this situation would be. A good approach is then to simply test a range of noise values and look at the results, to have an indication what the neural network is able of handling.

This has been done for the standard data sets used before of data of spheres, horizontal cylinders and vertical cylinders, respectively. The gravity acceleration values were taken, and an uniformly distributed number between plus and minus a noise percentage times the norm. The noise percentages were 5%, 15% and 25% and the uniform distribution was chosen such that the percentages are more indicative of the noise; a percentage of 5% means no more than 5% noise. For this analysis, the 1D ANN's assuming the three different shapes have been used, as these give the best result in terms of distinction between depths and shapes in the output. Due to the size and similarity of the graphs, two of them (the one of 5% and 25%) have been put in Appendix in Figure 1 and Figure 2. Shown beneath in Figure 6.16 is the situation with a 15% noise value. For analysis, it is most insightful to compare this situation with the graph of Figure 6.15. For clarity, in Figure 6.16 more data points were used (1000x15 instead of 100x15), but since the boundaries of the distribution are quite clear, this does not hinder the comparison.

It can immediately be seen that the orange line, which had very good fits with the corresponding data sets in Figure 6.15, does not show the same behaviour in Figure 6.16, and when it does, it is not with the corresponding data set. This is understandable, as the ANN's are trained on data without noise, yet tested on noisy data. There is no match expected, and the matches which are present are erratic. What is also visible is that the point defined as 'inflection points' previously have shifted with respect to the situation in Figure 6.15. As it is unclear why they are present in the first place, it is also not understood why they move, and not only in the horizontal but also the vertical direction. This makes the

figures of Figure 6.16 still comparable with the ones of Figure 6.15, but also noticeably different. Another very noticeable difference is that the spread of the data points in Figure 6.16 is much higher compared to the data points of Figure 6.15. This is understandable, as the data points are more varied, and the data points of a single batch belonging to a particular combination of depth and shape don't perfectly match this combination anymore. Also a neural network perfectly performing on earlier data would show spread in this case, as there is no perfect solution. Lastly, notice that in Figure 6.16 that all nine graphs are much more alike than the nine graphs of Figure 6.15. This is a sign that the ANN's have trouble with discerning differences between data from spheres and data from the cylinders. This is since the data is indeed not very different when noise is added; the noise can result in a data point from a spherical cavity fitting the situation with a horizontal cavity better. This situation makes it much more difficult to use this neural network, as the crux of this method is comparing graphs resulting from a data set to the graphs of Figure 6.16. These features are also visible in Figure 1 and Figure 2 in the appendix in a lesser or larger extent.

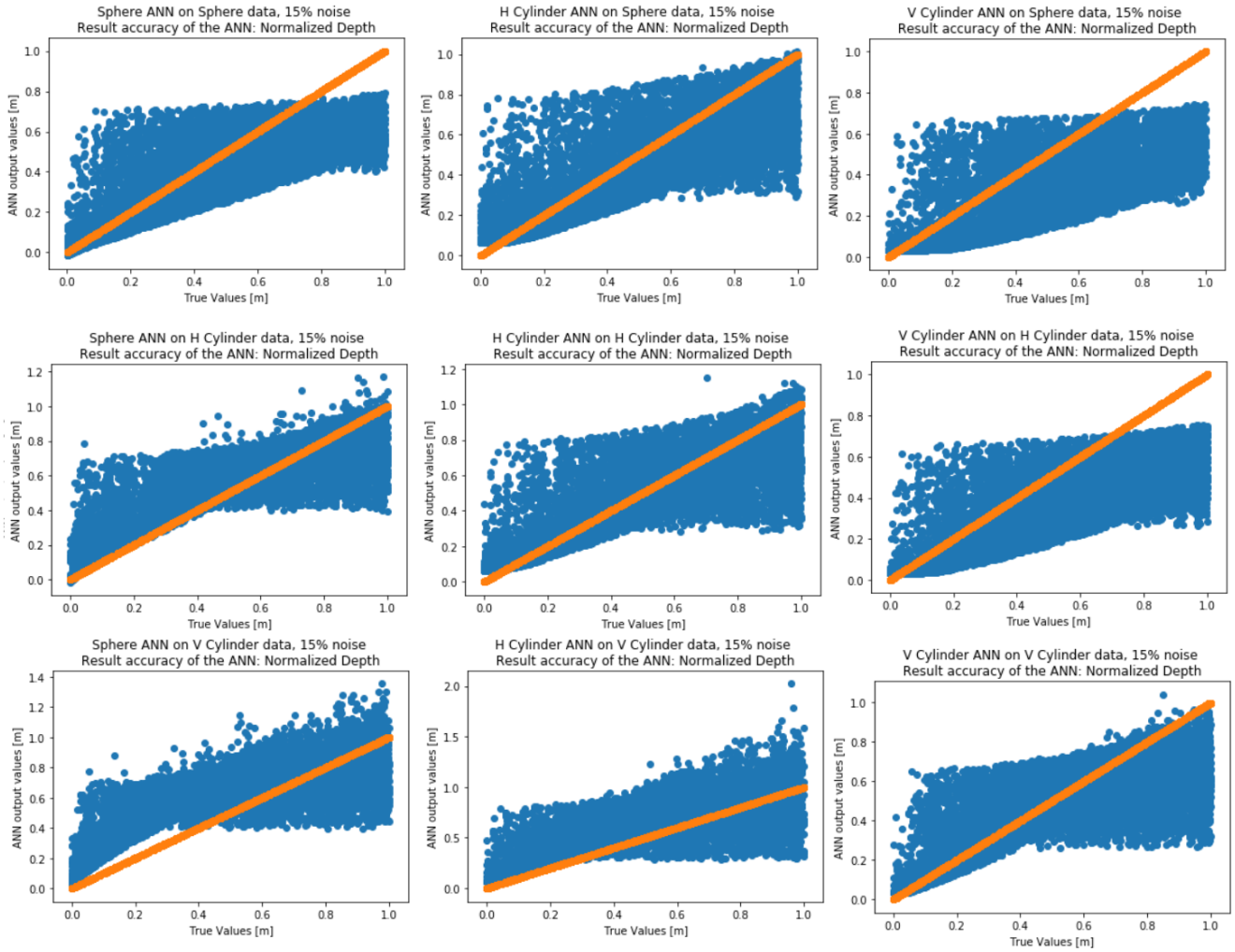


Figure 6.16: Results of usage of various data sets with 15% noise on various ANN's with default settings, each assuming a different shape factor. The orange line shows a perfect response.

Lastly, a test was performed where an ANN assuming spherical data was tested on noisy data from spherical cavities, rather than 'clean' data, and then tested on the same data sets as before. The results are shown in Figure 6.17 and should be compared to the first column of results in Figure 6.16. As can be seen, the results of Figure 6.17 are much more similar, which is not a good sign; it means the ANN is not able to see the difference between shapes. Thus, it is advised in this situation to train on clean data and test on noisy data (or data from field surveys).

In summary, it can be seen that the performance of the neural networks is certainly influenced by the amount of noise on the data, and that from a certain point on the results are not useful anymore. For some applications, this may lie at 15%, but 25% seems to be the limit for accepting any kind of result; when the data is even noisier, it can not be expected to give good results with these ANN's. This makes the method rather sensitive to noise.

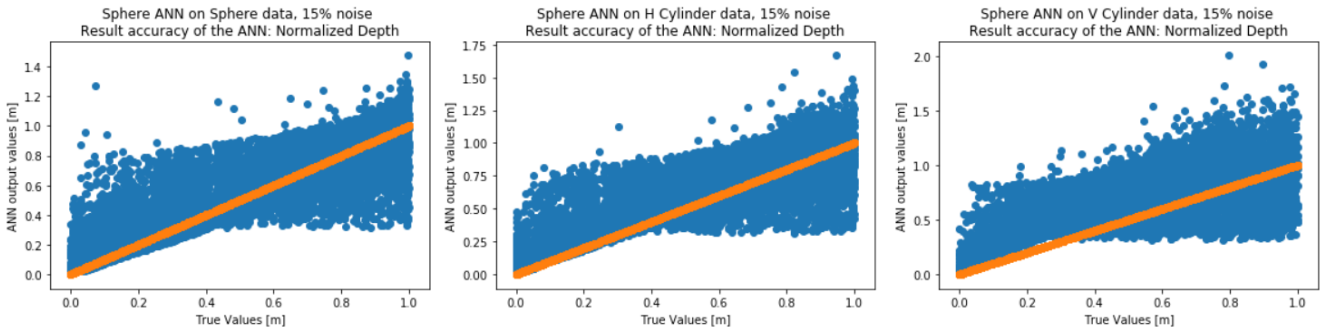


Figure 6.17: Results of usage of various data sets with 15% noise on the sphere ANN with default settings, where training was performed on a noisy data set The orange line shows a perfect response.

6.9. Cavity data

At last, the neural network is tested on realistic data. Data is obtained from modelling a lava tube of which the map is obtained by LIDAR investigation. The shape is far from an elementary shape, but loosely follows a horizontal cylindrical shape. Two depths are taken, which would not be the case in an actual survey, as there is only one configuration to test. For the ANN's assuming a sphere or vertical cylinder, the data is easy to transform; the maximum anomaly is looked for, and then points are selected and the distance to the maximum anomaly is taken. For the horizontal cylinder ANN, it is slightly different. As mentioned previously, there is not supposed to be only one location of maximum anomaly, but a line with identical gravity acceleration. While this will not be the case in real-life situations, one must still identify a general direction, and the distance of random points to this vector must be taken. This can be done by finding the argument of the vector of the general direction, subtract this from the argument of the random points, and then take the absolute value of the y-coordinate; it is simply a transformation. This has been done for the data and these data points are used as testing data. The three ANN's for the three shapes are tested on data coming from their respective shapes, and then tested on the cavity data. The results are shown in Figure 6.18, for clarity the actual depth values (40 meter and 80 meter, normalised) are also shown, though they are supposed to be unknown. These three graphs should now be compared to fig:1DResults, Figure 1, Figure 6.16 and Figure 2, and the amount of noise should be approximated. From a manual comparison with the results from elementary shapes, this is about 20% for the horizontal cylinder data and about 25% for the sphere and vertical cylinder data; the data fits the horizontal cylinder shape better. Yet, this indicates a comparison with Figure 2, and even with the depth value known it is almost impossible to state that indeed the data comes from a cavity with approximately a horizontal cylinder shape; it could be any shape. If not even the depth value is known, it is impossible to draw conclusions. The observations from Section 6.7 and Section 6.8 seem to hold; the neural network in its current state is not able to be applied in real-life use, though the results with this bare-bone ANN so far show that there is potential.

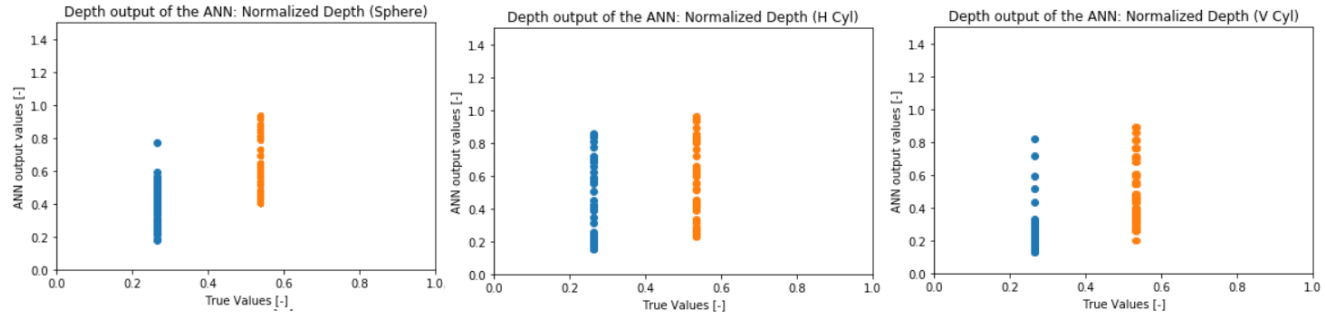


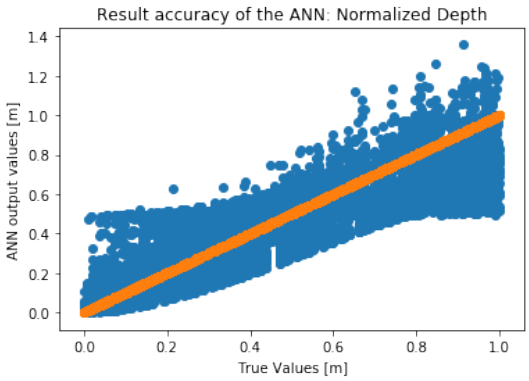
Figure 6.18: Results of training on perfect data and testing on cavity data for the three ANN's settings.

6.10. Comparison with supervised learning

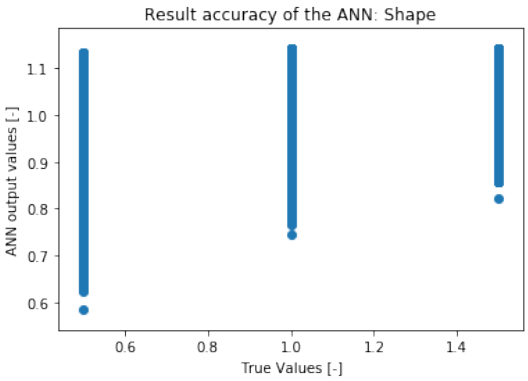
While it is not the goal of this report, it is still important to compare these results to the results which a supervised network would give, so one to which the true values for z and q are supplied during training. Of course, it is expected that this method gives (much) better results, as very relevant information is given. Yet, this method is not expected to be optimal in all cases, for example when real-life data is used which can not be categorised in a certain shape. However, for now it serves as a good comparison. Additionally, in this step it is possible to use different ANN's with entirely different structures, as it is much easier to use ANN templates in MatLab and Python for supervised learning compared to adapting one to be useful for self-supervised learning. For this step, the self-supervised network which was developed for this course

is copied and adapted for supervised learning, a neural network from MatLab's 'nntool' was used and a neural network framework from the Tensorflow package in Python has been used. The settings were identical, as far as this was possible; the MatLab and Python packages have many added features compared to the self-developed version, and automatically used the Levenberg-Marquadt algorithm rather than simple steepest descent for the back-propagation. Yet, it is also intriguing to see what kind of capabilities these methods then have.

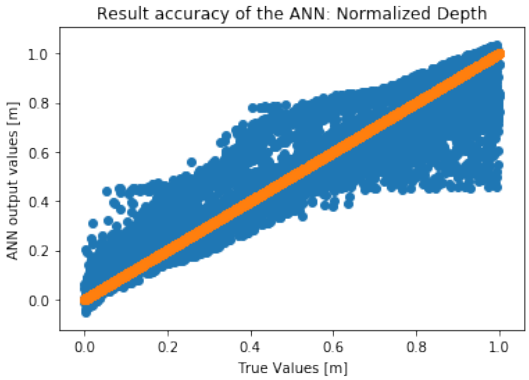
The results of this analysis are shown in Figure 6.19 with simple data without noise. There are a number of very remarkable results to point out. First of all, it is a very good sign that all ANN's, despite being based on entirely different frameworks, give very similar results. This also backs the argument that the self-developed neural network performs like expected and desired, despite it being fairly simple. Yet, it also shows that this is an average output of the supervised neural network, which is still far from perfect, shows quite some spread in the depth graphs, and most importantly is still unable to distinguish between shape factors, apart from very low output values. The main surprise is then that these results are not too different from the results of the basic self-supervised, self-developed two-dimensional neural network. The main difference is that in the case of the results from Figure 6.19, the orders of magnitude of both the depth and the shape are much closer to the expected values than for the results from the self-supervised neural network.



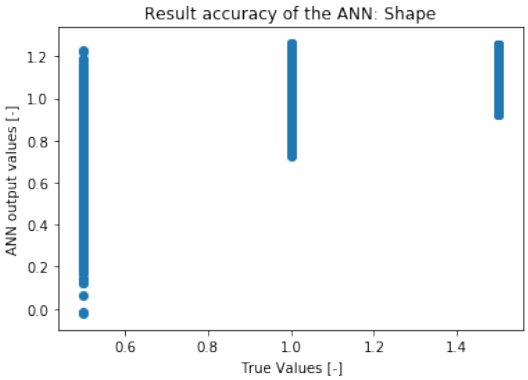
(a) The depth output of the supervised ANN when training data is used as input, self-developed variant



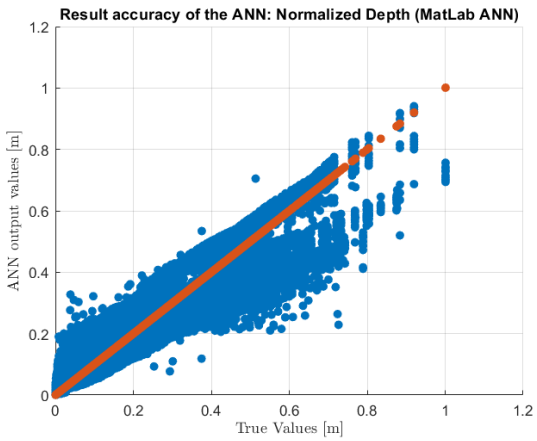
(b) The shape factor output of the supervised ANN when training data is used as input, self-developed variant



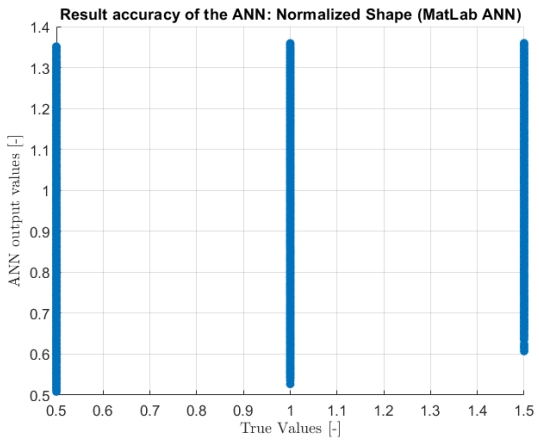
(c) The depth output of the supervised ANN when training data is used as input, Tensorflow-based variant



(d) The shape factor output of the supervised ANN when training data is used as input, Tensorflow-based variant



(e) The depth output of the supervised ANN when training data is used as input, MatLab-based variant



(f) The shape factor output of the supervised ANN when training data is used as input, MatLab-based variant

Figure 6.19: Depth and shape factor output for various supervised ANN's. The orange/red lines are perfect responses

The main question is of course why this is the case. For many neural networks and problems, a distribution of output values very close to the orange line can be expected. Yet, even with established methods and frameworks, this is not achieved. It is expected that the multi-dimensionality of both input and most importantly output gives problems regarding constraints and output values that match all input, as this can be the case for other problems as well [4]. Another option is that the data itself is too difficult to interpret for the neural network, as there might be many similarities between data from a spherical cavity and a horizontal cylindrical cavity. The many reports and articles about the use of neural networks for gravity data interpretation suggest otherwise, but it is not exactly known how they have set up their neural networks. Yet, at this point it falls beyond the scope of this project to also investigate this issue, and the conclusion is that the self-supervised network performs on a very comparable level compared to the supervised neural networks.

7

Conclusion

The aim of this project was to investigate how self-supervised artificial neural networks could aid in the interpretation of gravity data obtained from field surveys, specifically for lunar environments. The ANN could help identifying the depth and shape of buried tunnels and caves, as Equation 4.4 helps with classification of caves into simple parameters while still maintaining a multi-dimensionality fitting for a complex problem. Normalisation of this equation into Equation 4.6 resulted in a formula which could be used for self-supervised learning, as this included the four most important variables, two of them measurements, to of them cave parameters. With modifications on the basic back-propagation algorithm for neural network training, an alternative loss function was found in Equation 4.11; the needed derivatives to the relevant parameters are more complex than for the traditional supervised methods, yet equally applicable, provided the outputs are not (close to) zero.

In order to learn more about the basics of neural networks and how they work, I programmed a neural network from scratch in Python. While this took more time than using an existing framework, it also made it possible to understand better what can be changed when a neural network is not behaving as expected, and it gave a crucial understanding of the different steps of the neural network. During building and testing, it was already found that the problem to tackle is challenging for the artificial neural network, possibly due to the derivative having very sharp gradients in some areas of the parameter space. Yet, the development was continued with this self-developed neural network, mostly influenced by the fact that it turned out to be more difficult than expected to adapt existing neural network frameworks in terms of their loss functions, especially when this is also dependent on neural network parameters. The conclusion was that this required more knowledge and time than available. Yet, it was also concluded that investigating a mediocre-performing neural network can give more insight in the influence of settings than a well-performing one.

The most important settings which were varied are the number of hidden neurons, the initial learning rate, the number of training epochs and the size of the data set. Metrics used for this analysis are the loss at the final epoch as well as the spread in the final results; batches of data coming from the same configuration were always used. With this, all inputs of the same batch should result in the same output. As this is not the case, the spread in these results is a metric of the accuracy. Variation in these metrics has been found, for example in Figure 6.1, but in general results like in Figure 6.2 were found, with a clear triangle or seesaw shape for the depth value output and a figure like Figure 6.5 for the shape factor. This was even more peculiar with the tests for large data sets, for example in Figure 6.11a, where the boundaries of the result distribution are clear and sharp.

In order to understand this behaviour better, other settings were changed as well. Limited change could be performed on the loss function due to the self-supervised nature, but two alternatives could be found, with results in Figure 6.13 and Figure 6.14, again showing similar results as previously seen. The idea was that this had something to do with the multi-dimensionality of the problem, so the choice was made to make three separate ANN's, each for one specific shape factor. Despite limitations, this gave more useful results, as showed in Figure 6.15. For a sensitivity test, artificial noise was added to the data and the ANN's were tested on this, which showed that the usefulness of the previous results are limited for higher noise levels. This was also visible when testing on actual cavity data, as in Figure 6.18, and the neural networks are not on such a level that they can be used for real-life data analysis. Surprisingly enough, this is not fully due to the fact that it is self-supervised instead of supervised, or because it has been self-developed; the results of Figure 6.19 show of supervised learning in different frameworks suggest that the issue lies either in the problem of data being too similar, or the multi-dimensionality of the problem.

In conclusion, to the knowledge of the author no research has been done on self-supervised learning in artificial neural networks for gravity data interpretation. This project showed that there is potential and that self-supervised neural networks can perform on similar levels as supervised neural networks, but that the current implementation encounters too many challenges when being applied to real-life data to be considered useful.

8

Discussion

While results have clearly showed that (self-supervised) neural networks can provide an independent check of results obtained from other sources, it works far from perfect and is not fit for real-life use. It is expected that this either is because of the two-dimensionality and Equation 4.6 being under-determined for two inputs alone, or because the data is too much alike to be classified into different shapes. If it is the dimensionality, the learning algorithm and the loss function should be investigated in more detail, as so far all used networks have had issues with getting the final loss value very low, and all plots show a lot of spread. Evidently, the central problem of this investigation is not fit for the usual standard neural network frameworks, and requires a more specialised neural network, which could be the subject for further research. While it is possible that the data itself is too difficult to handle, the wide variety of (supervised) ANN's of previous researches shows that there is more potential in the use of them which has not been found in this report.

There are a number of recommendations for future work on this. Regarding data generation, more data should be obtained from actual gravity modelling outputs, as these are the result of realistic settings. So far, limited use has been made of this, also since the neural networks would not be able to perform well on this. However, to judge whether they can be used on a potential lunar mission, this is a crucial step to make. Additionally, lava tubes with a 'skylight' should also be modelled; these were not available for this research, as they are rare on Earth. It is also very much advised to make the gravity modelling process automatic, which was also not the case during this project.

A more model-related recommendation is to test the performance also with multiple layers and different training algorithms, things that have proven to be too difficult to implement in the time frame of the project.

There is an important note to make regarding cavities with the shape of a horizontal cylinder. For this, rather than identifying the position with the largest gravity anomaly, a general direction must be identified. This can also be automated, but at the same time introduces an error. Most importantly however, if a clear direction in the gravity anomaly can be seen, it is already expected that the cavity has the shape of a horizontal cylinder, or is in any case horizontally elongated. On the contrary, spherical cavities and those in the shape of a vertical cylinder are radially symmetric and don't show this behaviour.

Lastly, this again shows that a neural network can never be expected to have the full task of gravity data interpretation, and either manual checks have to be done, or preferably established gravity inversion techniques should be used. Yet, due to the property of the ANN requiring very little computational power once trained and that it gives an independent check on other inversion results, the potential of the use of an ANN in the context of the mission concept is not to be forgotten.

Bibliography

[1] D. M. Hurwitz, J. W. Head, and H. Hiesinger, *Lunar sinuous rilles: Distribution, characteristics, and implications for their origin,* Planetary and Space Science **79**, 1 (2013).

[2] G. Colley, *The detection of caves by gravity measurements,* Geophysical prospecting **11**, 1 (1963).

[3] H. Saibi, M. Amrouche, and A.-R. Fowler, *Deep cavity systems detection in al-ain city, uae, based on gravity surveys inversion,* Journal of Asian Earth Sciences **182**, 103937 (2019).

[4] A. Hajian and P. Styles, *Application of soft computing and intelligent methods in geophysics* (Springer, 2018).

[5] E. D. General, *System studies - lunar caves,* (2020).

[6] A. A. Grêt, E. E. Klingelé, and H.-G. Kahle, *Application of artificial neural networks for gravity interpretation in two dimensions: a test study,* Bollettino di Geofisica Teorica e Applicata **41**, 1 (2000).

[7] A. Salem, E. Elawadi, and K. Ushijima, *Depth determination from residual gravity anomaly data using a simple formula,* Computers & geosciences **29**, 801 (2003).

[8] A. Hajian, H. Zomorrodian, and P. Styles, *Simultaneous estimation of shape factor and depth of subsurface cavities from residual gravity anomalies using feed-forward back-propagation neural networks,* Acta Geophysica **60**, 1043 (2012).

[9] A. Eshaghzadeh and A. Hajian, *2d inverse modeling of residual gravity anomalies from simple geometric shapes using modular feed-forward neural network,* Annals of geophysics **61**, 115 (2018).

[10] T. Kaku, J. Haruyama, W. Miyake, A. Kumamoto, K. Ishiyama, T. Nishibori, K. Yamamoto, S. T. Crites, T. Michikami,

Y. Yokota, *et al.*, *Detection of intact lava tubes at marius hills on the moon by selene (kaguya) lunar radar sounder,* Geophysical Research Letters **44**, 10 (2017).
[11] Q. Huang, Z. Xiao, and L. Xiao, *Subsurface structures of large volcanic complexes on the nearside of the moon: A view from grail gravity,* Icarus **243**, 48 (2014).
[12] L. Chappaz, R. Sood, H. J. Melosh, K. C. Howell, D. M. Blair, C. Milbury, and M. T. Zuber, *Evidence of large empty lava tubes on the moon using grail gravity,* Geophysical Research Letters **44**, 105 (2017).
[13] A. N. Deutsch, G. A. Neumann, J. W. Head, and L. Wilson, *Grail-identified gravity anomalies in oceanus procellarum: Insight into subsurface impact and magmatic structures on the moon,* Icarus **331**, 192 (2019).
[14] B. A. Campbell, B. Hawke, and D. B. Campbell, *Surface morphology of domes in the marius hills and mons rümker regions of the moon from earth-based radar data,* Journal of Geophysical Research: Planets **114** (2009), https://doi.org/10.1029/2008JE003253.
[15] P. Lee, *Ice-rich caves on the moon and mars: Prospects and pragmatic recommendations for exploration,* Third International Planetary Caves Conference, San Antonio, Texas **2197**, 1066 (2020).
[16] J. Haruyama, K. Hioki, M. Shirao, T. Morota, H. Hiesinger, C. H. van der Bogert, H. Miyamoto, A. Iwasaki, Y. Yokota, M. Ohtake, *et al.*, *Possible lunar lava tube skylight observed by selene cameras,* Geophysical Research Letters **36** (2009), https://doi.org/10.1029/2009GL040635.
[17] A. K. Theinat, A. Modiriasari, A. Bobet, H. J. Melosh, S. J. Dyke, J. Ramirez, A. Maghareh, and D. Gomez, *Lunar lava tubes: Morphology to structural stability,* Icarus **338**, 113442 (2020).
[18] L. Bessone, I. Carnelli, M. Fontaine, and F. Sauro, *Esa sysnove lunar caves challenge,* Third International Planetary Caves Conference, San Antonio, Texas **2197**, 1039 (2020).
[19] D. K. Butler, *Microgravimetric and gravity gradient techniques for detection of subsurface cavities,* Geophysics **49**, 1084 (1984).
[20] J. Friedrich, C. Gerstenecker, and O. Gürkan, *Gravimetric examination of hagia sophia's subsurface structure,* Journal of Geodesy **70**, 645 (1996).
[21] T. Mochales, A. Casas, E. Pueyo, O. Pueyo, M. Román, A. Pocoví, M. Soriano, and D. Ansón, *Detection of underground cavities by combining gravity, magnetic and ground penetrating radar surveys: a case study from the zaragoza area, ne spain,* Environmental Geology **53**, 1067 (2008).
[22] A. R. B. Harun and A. R. Samsudin, *Application of gravity survey for geological mapping and cavity detection: Malaysian case studies,* Electronic Journal of Geotechnical Engineering **19**, 8247 (2014).
[23] R. Kirsch, *Groundwater geophysics; A tool for hydrogeology*, Vol. 493 (Springer, 2006) pp. 319–345.
[24] B. J. Lyndon, *Apollo 17: Preliminary science report* (National Aeronautics and Space Administration, 1973) pp. 298 – 310.
[25] E. Elawadi, A. Salem, and K. Ushijima, *Detection of cavities and tunnels from gravity data using a neural network,* Exploration Geophysics **32**, 204 (2001).
[26] G. Cybenko, *Approximation by superpositions of a sigmoidal function,* Mathematics of control, signals and systems **2**, 303 (1989).
[27] G. de Croon and E. van Kampen, *Artificial neural networks and self-supervised learning,* University Lecture, Delft University of Technology (May 2020).
[28] H. N. Gharti, J. Tromp, and S. Zampini, *Spectral-infinite-element simulations of gravity anomalies,* Geophysical Journal International **215**, 1098 (2018).

APPENDIX

.1. CODE

All code for this project can be found in this link; due to the length of the codes, these are not copied here. A short description follows per script:

- 'ANN Gravity Signals - 1D variant - H Cylinder.ipynb': self-supervised ANN for 1-dimensional loss function; assumes all data comes from horizontal cylindrical cavities
- 'ANN Gravity Signals - 1D variant - Sphere.ipynb': self-supervised ANN for 1-dimensional loss function; assumes all data comes from spherical cavities
- 'ANN Gravity Signals - 1D variant - V Cylinder.ipynb': self-supervised ANN for 1-dimensional loss function; assumes all data comes from vertical cylindrical cavities
- 'ANN Gravity Signals - Data Generation.ipynb': script intended purely for data generation, but saw little use during the project; usually the data was generated in the ANN files themselves
- 'ANN Gravity Signals - SL.ipynb': supervised ANN for the standard 2-dimensional loss function
- 'ANN Gravity Signals - SSL Alternative Loss 1.ipynb': self-supervised ANN for an alternative 2-dimensional loss function, solved for x instead of g.
- 'ANN Gravity Signals - SSL.ipynb': self-supervised ANN for the standard 2-dimensional loss function, main script during this project
- 'ANN Gravity Signals - SSL Alternative Loss 2.ipynb': self-supervised ANN for an alternative 2-dimensional loss function, taking a sum instead of one value.
- 'ANN Gravity Signals - TensorFlow Self-Supervised ANN.ipynb': self-supervised ANN for the standard 2-dimensional loss function developed in the Tensorflow framework in python. Due to technical errors, this did not see use.
- 'ANN Gravity Signals - TensorFlow Supervised ANN.ipynb': supervised ANN for the standard 2-dimensional loss function developed in the Tensorflow framework in Python
- 'Cavity_data_for_ANN.m': script to extract data for the ANN from previosuly obtained results.
- 'ANN Gravity Signals - Working scripts.ipynb': script with various stages of the final ANN, showing the development and added features
- 'NNMatlab.m': script for generating data for the Matlab supervised ANN, which is used by calling the functionality 'nntool'
- 'PlotNNResults.m': script for plotting the results of the Matlab supervised ANN

.2. ADDITIONAL FIGURES

These figures are the large graphs consisting of 3x3 figures of the noise analysis of Section 6.8 of 5% noise and 25% noise.

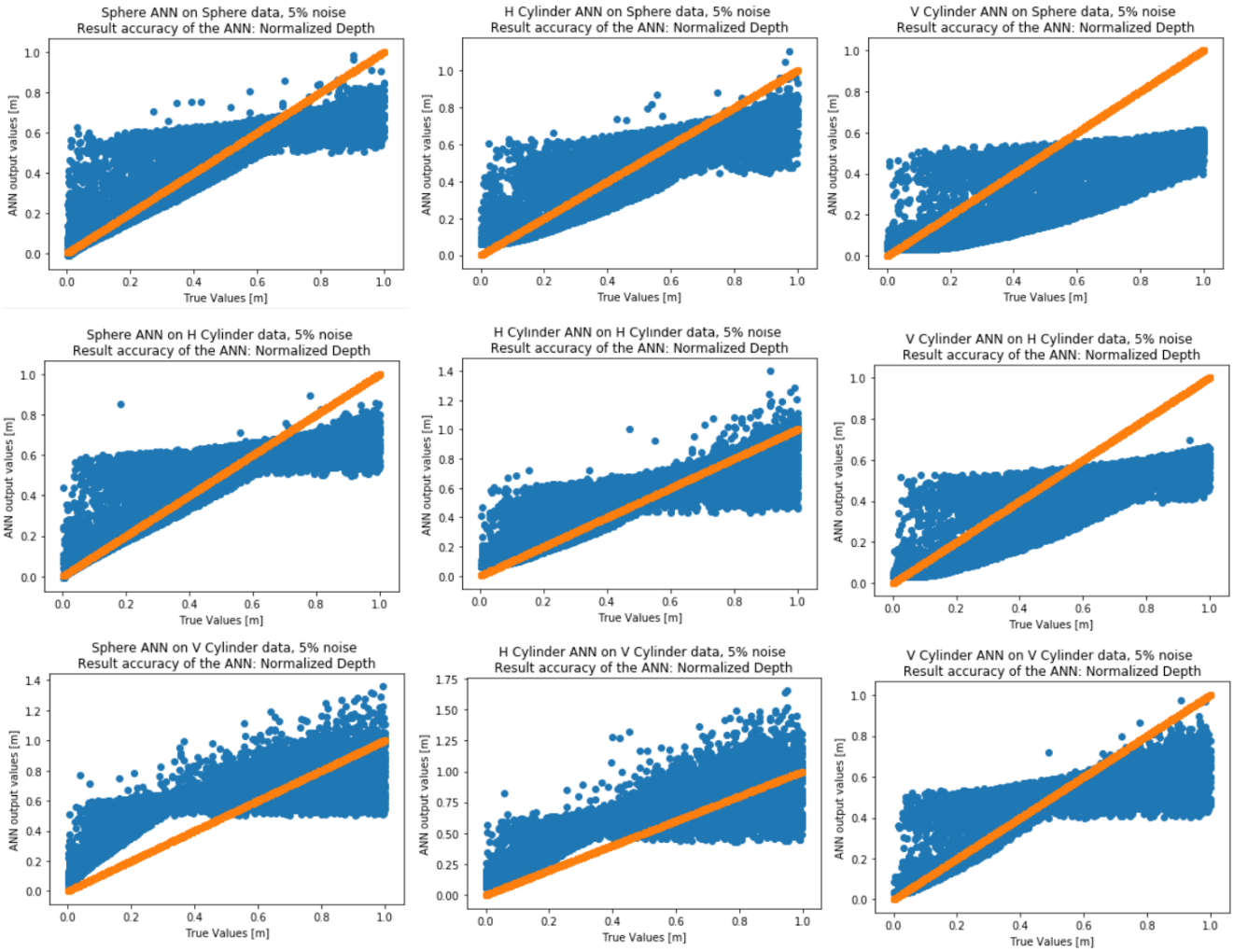


Figure 1: Results of usage of various data sets with 5% noise on various ANN's with default settings, each assuming a different shape factor. The orange line shows a perfect response.

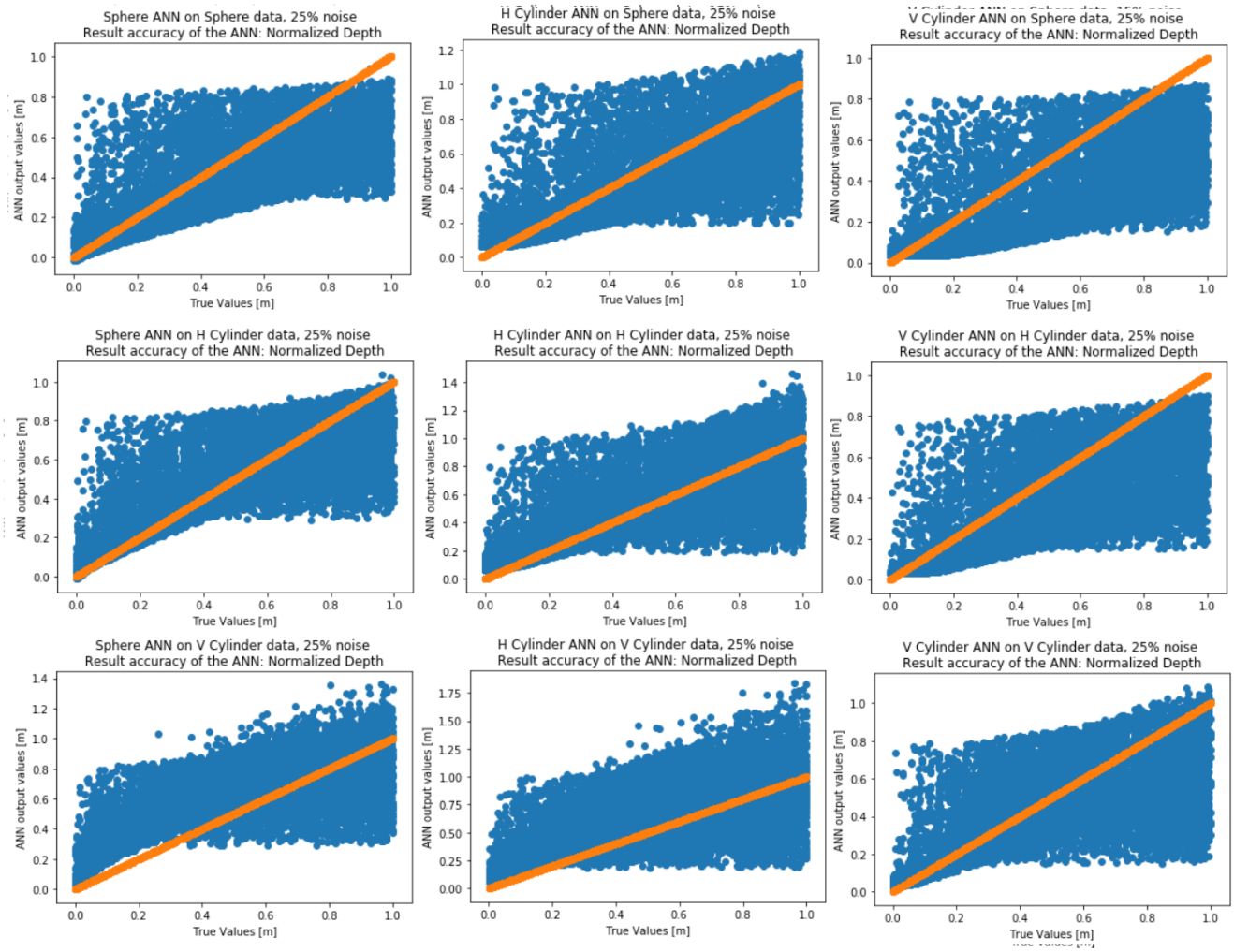


Figure 2: Results of usage of various data sets with 25% noise on various ANN's with default settings, each assuming a different shape factor. The orange line shows a perfect response.