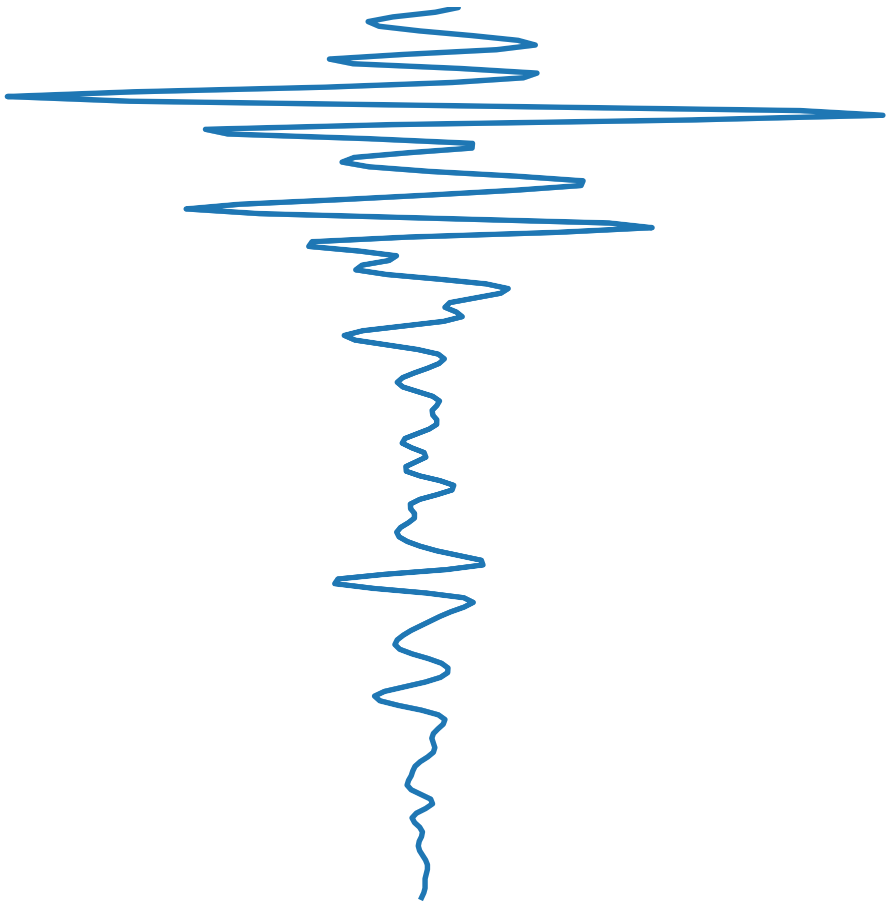
Rosalie Verwijs | August 2021

THESIS MSC. APPLIED GEOPHYSICS

Machine Learning Techniques for Moment Tensor Estimation



A Master Thesis for the Applied Geophysics program by the IDEA league:

Delft University of Technology
Swiss Federal Institute of Technology
Rheinisch-Westfälische Technische Hochschule Aachen

Examination Committee

Dr. Ir. F.C. Vossepoel,	TU Delft
Dr. E. Ruigrok,	Utrecht University, KNMI
H.A. Diab Montero,	TU Delft
Dr. Ir. D.S. Draganov,	TU Delft
Dr. C. Schmelzbach	ETH Zürich

the 6th of August, 2021

Abstract

As a result of the gas extraction in Groningen (the Netherlands) the amount of earthquakes have increased over the past decades. Understanding the induced seismicity in the Zeerijp area is important for the population living in this relatively densely populated region, but also essential to the Dutch governement and the operator of the field; the Nederlandse Aardolie Maatschappij (NAM). One way of gaining the knowledge on the dynamics of the fault systems in the area is by, for example, a full waveform inversion f recorded seismic data that gives us the moment tensor describing the earthquake mechanism. A drawback of this method is that it is computationally expensive and time intensive. In this thesis two different machine learning techniques are investigated that can lead to a faster estimation of the moment tensor. We find that, when using time traces that are modelled from the 1D velocity model of the area, there is a difference in sensitivity of the network to the individual moment tensor components. Besides that, using a feed forward neural network generally yields a better performance, but does not give us the associated uncertainties of the network. This is in contrast with the mixture density network, which performs slightly worse than the feed forward network, but it does give us the involved uncertainties when the algorithm makes the predictions. Adding different levels of Gaussian noise to the data gave us a first insight as to how the precision of the moment tensor estimations would change. It was found that a mixture density network has a more stable prediction when estimating low signal-to-noise ratio data. No test with real data from the Zeerijp area is done, however, we can conclude from our results that the estimation of the moment tensors in this region should be possible with the use of machine learning techniques.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

With the size of nine hundred square kilometers, the Groningen Gas field is the largest on shore natural gas field in Europe. Situated in the Netherlands, it occupies almost a third of a province where around half a million people live. In earlier times, when the gas field was discovered, it created a lot of job opportunities for the inhabitants in the north of the Netherlands. Next to that it caused the rapid transition from coal to the much cleaner gas for a substantial percentage of the dutch households ((Historiek 2019)). All seemed good, until the nineties, when small earthquakes started occurring in the area of the Groningen gas field. Slowly, but steadily, the amount of earthquakes per year increased, see Figure 1.1. The government did not immediately take action, until the earthquake in Huizinge. This event happened in 2012 and had a magnitude of 3.6, which was the last straw for the province of Groningen. It was also after this event that the already existing network of seismometers that monitored the area of the gas field was expanded with 69 new stations. These stations have different levels of sensors in order to increase the hypocentre accuracy and to give a better understanding of the source mechanism (Dost et al. 2017). And it is this last objective that is of particular interest to us.

So what makes investigating the Groningen area so interesting? First of all, it is important to realize that when the field was discovered, it was the biggest onshore gas field in the world. Nowadays, the field still ranks among the top ten, however, it is a very well researched area since it was the biggest for quite a time. Another point of interest is, is that with the decline of the gas production, one has come to realize that the field is much more complex than initially thought (de Jager & Visser 2017). Add this complexity up to the fact that the gas field is situated in a (especially compared to larger onshore gas fields) urban area, it makes for a pretty interesting case. Even though a majority of the events taking place in this area belong to the category of micro earthquakes, i.e., earthquakes with a magnitude smaller than 2.0, some of these events can cause significant damage. Previous research on the Groningen Gas Field on how to monitor and better understand the seismicity in this area has been done. As mentioned earlier, the expansion of the seismic network can lead to more knowledge on the source mechanism of each event that is taking place.

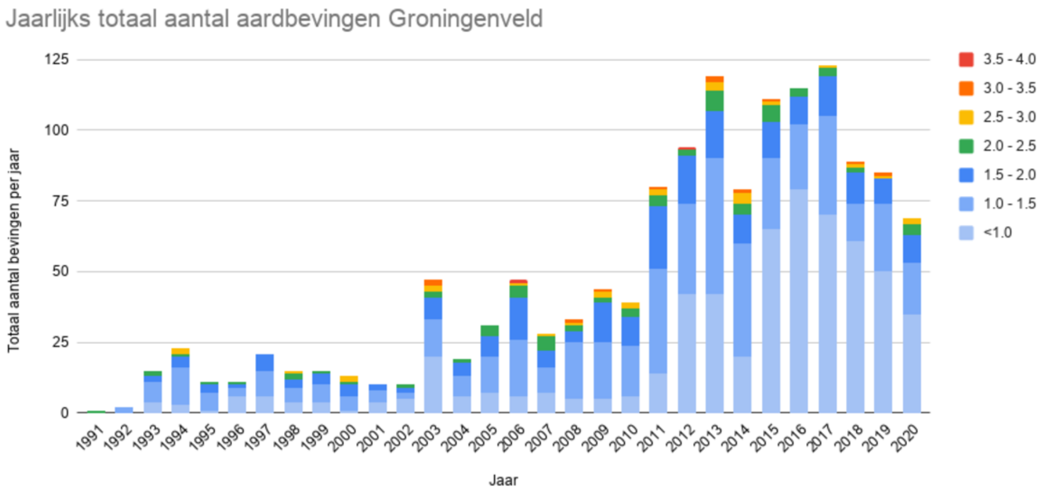


Figure 1.1: Number of earthquakes in the Area of the Groningen Gas Field. On the horizontal axis the year is displayed, the vertical axis shows the amount of earthquakes, and the color indicates the magnitude of the events. Source: (KNMI 2020)

The source mechanism tells us the fault displacement during an earthquake. One of the ways of representing the source mechanism is by the moment tensor. To understand this, it is first important to know how an earthquake can exist in the first place. See it like this; Before the event happens, the rocks surrounding the fault will experience a significant increase in shear stress. The stress will cause a rise in elastic strain energy. In the area of the Groningen Gas Field there already are faults present in the underground. Meaning that if the stored energy overcomes the friction on the already existing faults, an earthquake will occur. The two parts of rock slide along the fault and the stored energy will be released in the form of for example seismic waves. The amount of energy released is related to the magnitude of the event; the more energy, the higher the magnitude. The moment tensor is a representation of the forces acting on the system at hand and can be derived from a seismogram. For example: the first inversions of the moment tensor proved to be computationally expensive (Willacy et al. 2019*a*). By using a full-waveform probabilistic optimization and the use of the Pyrocko package (Developers n.d.), (Dost et al. 2020) made it possible to retrieve the source mechanism (by means of the moment tensor) of thirteen events with a magnitude bigger than 2.0. Despite having quite some knowledge already on the specific area in Groningen, there is still an important part missing. Most commonly these source mechanism inversions are executed for larger magnitude events. This is done, because the signals are more clear, i.e. there is less noise visible in the data, which will make the result of the inversion more reliable.

The case of the Groningen gas field is a little bit different however. Here, we are not dealing with natural seismicity, as described above, but with induced seismicity. By taking out the gas, the pore pressure in the rock declines, leading to compaction. This is called poroelastic stressing and describes how the changes in stress caused by the declined fluid pressure can cause deformations of the rock, in this case the compaction of the reservoir (Buijze et al. 2017). The already present natural faults located within and around the reservoir are reactivated, resulting in the induced seismicity (Bourne et al. 2014). When the build-up stress overcomes the friction, kinetic-, heat-, and seismic wave energy are released. These three together form the earthquake. The seismic waves (body- and surface waves) travel through the earth and it is this energy that is recorded by the stations in the area. In order to get a better understanding of the induced seismicity, it could be valuable to study the micro earthquakes as well. For example, in their study, (Eyre et al. 2019), clearly show that the investigation of micro earthquakes could lead to a better prediction of larger, more damaging events. This is easier said then done, because the geology of the Groningen Gas field area is pretty complex, yet again showing why this problem is so interesting.

As some sort of follow up on the paper by (Dost et al. 2020) what we would like to do is find a way that makes estimating the source mechanism of each event in the Groningen Gas field area easier and computationally less expensive. As is mentioned by (Dost et al. 2020) is that their work was already an attempt to decrease the computational cost of finding the source mechanism of the events in Groningen. Instead of using finite difference modeling (as done by (Willacy et al. 2019*b*)) they proposed using a full-waveform probabilistic optimization method. For this method they only use the vertical and transverse components to predict the location, depth, and moment tensor solution of several events. The results give a detailed fault-slip model, however, the means of obtaining this accurate model takes time and a lot of parameter testing. This thesis will look at the possibilities of finding the source mechanism by the means of a neural network. Neural networks have become increasingly popular over the past years, since they provide a computationally lighter means of doing, for example, simulations that would normally have been a lot more intensive. Estimation of the moment tensor using a machine learning technique has been done before, successfully, but all using different techniques. For example (Binder 2018) shows that using a simple feed forward neural network, the moment tensors of microseismic synthetic data modelled using a simple velocity model, can be accurately predicted. (Käufl 2015) uses a mixture density network to not only predict the moment tensor, but also fault location and the magnitude of events in different regions in California. (Kuang et al. 2021) proof that with a more complex network (their own defined FMNet) the focal mechanism of the earthquakes in the Ridgecrest area with magnitudes greater than 5.5 can be predicted accurately. (Steinberg et al. 2021) takes this even further by combining a convolutional neural network with the theory of mixture density networks and test the performance of their algorithm also on the Ridgecrest data, but only using events that have magnitudes between Mw 2.7 and 5.5. These studies illustrate the complexity of the use of machine learning for moment tensor inversion, and form a good starting point of this thesis. The main goal of this study is as follows:

What architecture and which type of neural network can predict the moment tensor of the earthquakes in the Zeerijp region of the Groningen Gas Field area? And how can we assess the precission of these results?

In Chapter 2 the Zeerijp Area will be introduced. Zeerijp is only a small part of the area of the Groningen Gas Field, however, this particular region is of interest to us, because of the larger faults and the high seismicity. Here, we will also talk about the geology in the area, and why we can already predict how this might affect the performance of our neural network later on (Section 2.1). The second part of the same chapter, Section 2.3, will give a short review of how we will define the source mechanism of each event. First, we will talk about strike, dip, and slip, after which we will see that, in order to get a better representation of the source mechanism of an earthquake, it is much more convenient to use the moment tensor representation. Since the ultimate goal is to see if we can apply machine learning techniques to the data from the Zeerijp area, it is good to first check if the data in this area can be considered good enough for the network to even distinguish the independent moment tensor components. We can check this by looking at the signal-to-noise ratio in the area (Section 2.2). The second Chapter will dive into the methods used during this research. It will start with a look at how a change in source mechanism changes the response of the receivers measuring each event. Section 3.1 compares the different velocity models, source specific-, and receiver specific settings when a slight change in the source mechanism is made. Furthermore it will underpin the choices that have been made when making the different synthetic data sets, elaborating the importance of starting simple, then slowly building up to more complex data sets. After analysing the synthetic data an intro to neural networks will be given, especially the feed forward type of neural network. We will look at a standard classification case and expand this idea to regression problems. However, it is also in this section (Section 3.2) that we will find out why a conventional, i.e., a feed forward neural network will not completely suffice for our goals. Seeing that a conventional neural network only predicts the output values, there is no way of checking the uncertainties that the network faces when making its predictions. To evaluate the confidence of the algorithm we can switch to a different type of networks, namely mixture density networks. The theory behind this specific type of network and the differences with a more conventional network will be explained in Section 3.2.3. Please note that this thesis mainly focuses on the use of feed forward networks, and not, for example, convolutional neural networks, or other types. It is not that these types of networks could not achieve good results as well, on the contrary, take for example the paper by (Kuang et al. 2021), or (Steinberg et al. 2021), however, due to the time limit of this thesis it was decided to stick to feed forward types of networks. The results of both the feed forward- and mixture density networks are shown and discussed in Chapter 4. Following these outcomes a Discussion on the previous chapter is made, then the Conclusion will shortly summarize the findings of this thesis, and the Outlook will give some recommendations for future research.

Chapter 2

Theoretical Background

The Zeerijp area is a region in the province of Groningen (northern Netherlands) where a major part of the earthquakes caused by the Groningen Gas field takes place. It is vital to understand the geology and the importance of investigating this area to also understand the choices later made in this thesis. Following this a short analysis of the signal-to-noise ratio in the Zeerijp area will be given. This is done to investigate if the actual Zeerijp data will be of good enough quality to be analyzed by a neural network. After this introduction to the Zeerijp area and its real data, a refresher on the moment tensor, green's functions, and some of the usual ways to obtain these moment tensors is given.

2.1 The Zeerijp Area

Zeerijp is a small region in the province of Groningen in the northern part of the Netherlands. The area of interest is a mere 56 square kilometers, however, the geophone coverage is pretty high. As can be seen in Figure 2.1 the amount of stations covering a region of approximately 600 square kilometers sums up to thirty-seven. This came to be due to the big event in Huizinge in 2012. Since then the network has been expanded, nowadays containing seventy-nine stations in total. As said, Figure 2.1 shows the Zeerijp Area, bounded by the black box. The inverted orange triangles denote the borehole stations that each contain four receivers at different depths with intervals of fifty meters in between them. The blue lines outline the faults in the region, and it is these lines that cause trouble. As was explained in the Introduction a result of the gas production in Groningen is the induced seismicity in the area. Here, the majority of the faults are normal faults. Since the objective of this thesis is to investigate how well we can estimate the faulting mechanism from the response that we measure each time an earthquake takes place, it is important to also understand the geology of the area, because the response of the earth is formed by the geology in the surrounding region.

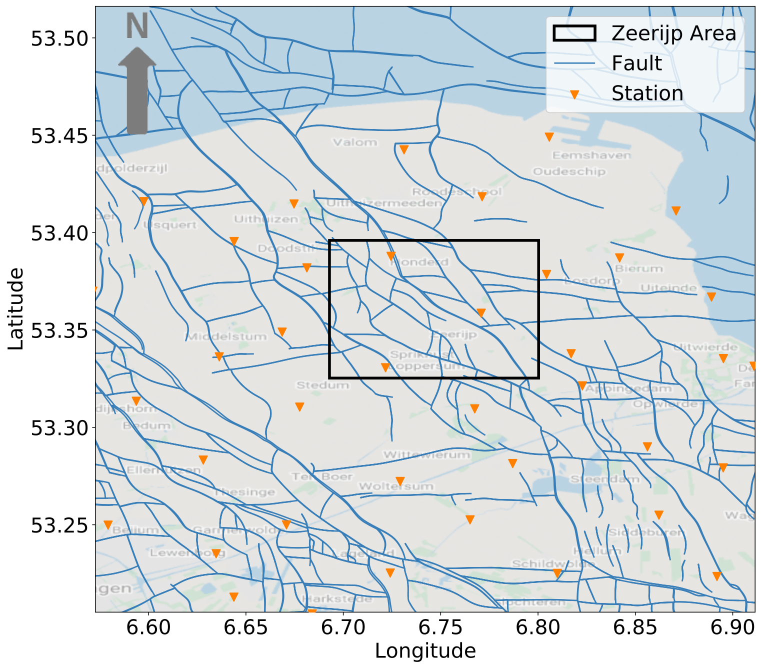


Figure 2.1: The Zeerijp area (black box) and all stations (inverted light orange triangles) within a range of fifteen kilometers of the borders of the region of interest. The light blue lines show the faults in the area, the grey and blue background display land and sea respectively.

The sediments that are related to the the Groningen gas field (Upper Permian Rotliegend) are deposited in a very big basin that stretches from UK to Poland. This Upper Rotliegend group shows itself as a sequence of red beds in the dutch underground. During the Carboniferous period (some 363 million years ago) major erosion took place in the region, resulting in temperature differences within the underground of the Groningen area itself that have not been explained (de Jager & Visser 2017). It was during this time also that coal seams were deposited. Following the period of erosion, rift basins started to form, something which has not been encountered in the rest of the Netherlands. These basins were filled with a mix of volcanic rock. Regional subsidence caused by thermal activity caused the deposition of the first Rotliegend sediments, following by the Upper Permian Zechstein and Lower Triasssic Buntsandstein sediments. During the Triassic period Pangaea started to break up and this led to a movement of the Zechstein salt. This lasted well into the Late Triassic period, forming big salt walls and domes, possibly triggered by the active faulting in the area at that time. The active faulting was driven even more by the further breaking up of Pangaea, resulting in the many fault blocks that can now be observed in seismic data (de Jager & Visser 2017). Moving into the Late Jurassic to Early Cretaceous period, it is believed that the initial development of gas took place during this time, which also denotes the main time period where all pre-existing faults were reactivated. However, as (de Jager & Visser 2017) state it in their research, the complex history of the Groningen Gas field area makes it hard to exactly say how and when the reservoir was actually charged. Apart from this, the initial gas pressures are fairly constant over the entire field. The sealing of the gas is formed by the Zechstein salt, closing off fractures and faults.

When the field was first discovered, the reservoir was called the Slochtern formation, named after the discovering well. Overlaying this reservoir rock are the Ten Boer and Ameland Memeber claystone layers. These layers form a barrier to the gas flow outside the Slochtern formation. From the coal seams formed during the Carboniferous time the gas has moved up and is being contained by the thick salt and anhydrite Zeichstein layers, forming the seal that traps the gas. An overview of the main tectonic phases can be found in Figure 2.2.

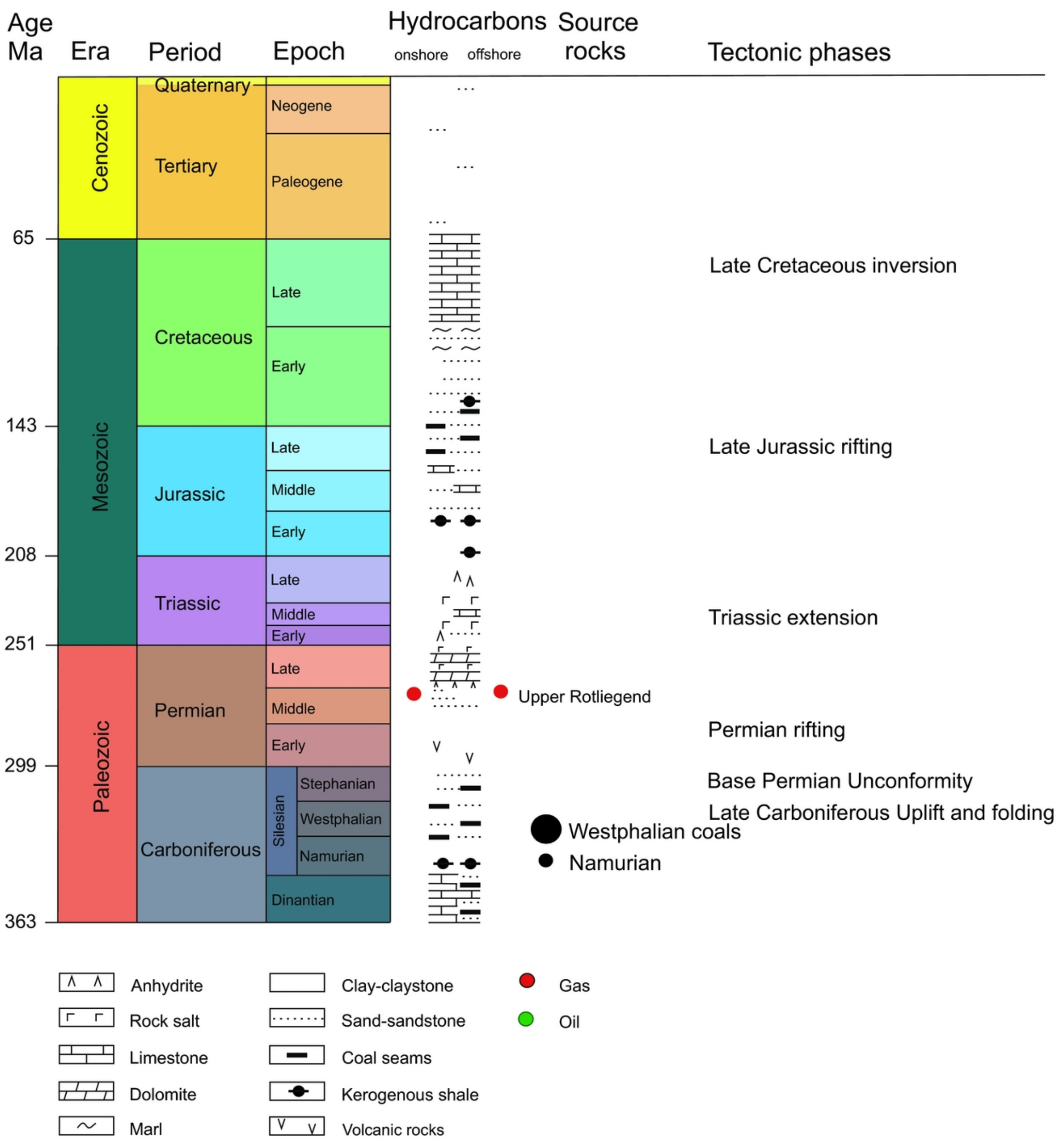
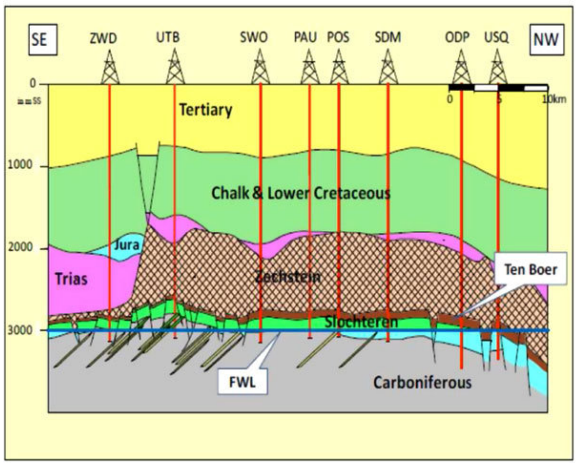
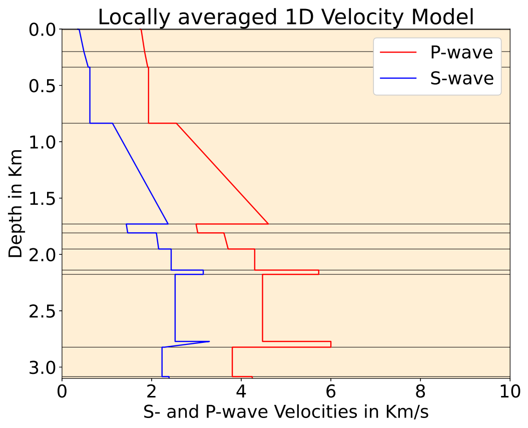


Figure 2.2: An overview of the nature and timing of several tectonic phases that formed the Slochtern formation, by (de Jager & Visser 2017).

Now that we know the tectonic history of the area of the Groningen Gas Field, we can look more into the geology of the Groningen Gas Field. Figure 2.3a shows that the Slochtern formation is mainly situated at a depth of 3000 meters. Above this we can spot the Ten Boer layer, and overlying the clay is the thick Zechstein formation. It is this layer that makes the Zeerijp area so interesting to us. The Zeerijp area is a region that needs to be monitored closely, because of the large population living at the surface. Be that as it may, another reason for this region being so interesting is the presence of the Zechstein formation. The imaging of salts is a very though process; often they are associated with weak reflections, forming some kind of blind spot into our seismic sections. But, the presence of a salt layer can hint at a possible reservoir trap, which was the initial impulse of boring wells in the province of Groningen. This brings us back to the increasing earthquakes due to the induced seismicity triggered by the pressure changes in the subsurface and the importance of understanding the faulting mechanism better. It is namely this Zechstein formation that makes it all a bit harder. Salt is notorious for deteriorating seismic signals in the earth. This is due to the high velocity contrast and impedance that can be found at the boundary between the salt and surrounding rock, which in turn causes very complex travel paths of the seismic waves. The same applies to the response of the earth that is picked up by the borehole stations over the area when an earthquake occurs. An example of the increasing complexity is shown in more detail in Chapter 3.1. The velocity model used for this



(a) Geological Cross-Section



(b) 1D Velocity Model

Figure 2.3: A cross-section of the geology in the Groningen Gas Field Area from South-East (SE) to North-West(NW) (Figure 2.3a). The abbreviated wells at the surface indicate the locations of these wells along the cross-section. As is seen is that the reservoir rock is mostly located around the 3000 meter mark in the subsurface. Just above the Slochtern formation we can find the Ten Boer clay layer. Overlaying the clay is the thick layer of Zechstein salts and anhydrites. By: (Vlek 2018). This cross-section can be compared to the velocity model in this thesis (Figure 2.3b). It is seen that the locally averaged 1D velocity model (Romijn 2017*a*) closely follows the geology of middle part of the geological cross-section.

thesis is based on the velocity model by (Romijn 2017*a*), i.e., we will be using a locally averaged 1D model. In his paper, (Romijn 2017*a*) describes how the Zechstein formation consists of so called anhydrite/carbonate floaters. These floaters change in thickness, some containing discontinuities, throughout the whole area. Since we will only use a locally averaged 1D velocity model, we cannot simulate the effect of these floaters into our data set, however, the multiple salt layers adding to the complexity will be visible in the velocity model.
A comparison of the velocity model to the cross-section is shown in Figure 2.3a. Here, we can clearly see the thick Chalk and Lower cretaceous layer, the thin Trias layer around 2000m depths, and the big Zechstein formation from 2000m to 3000m depth. The reservoir (indicated by the bright green layer in Figure 2.3a) with the overlying Ten Boer layer can also be distinguished in the locally average 1D velocity model.

2.2 Signal to Noise Ratio

The first question that might arise when we start investigating the Zeerijp area is; will the data from this area actually be 'good' enough to be able to use it for our goal, namely, estimating the moment tensor? One way to find this out is by looking at the signal-to-noise ratio (SNR). This ratio compares the level of a desired signal to the amount of noise in a trace. Noise is evidently present in any trace we have of the Zeerijp area. It can be caused by cars driving past, the motion of the waves from the coast that is close by, or animals rambling around in the fields. Obtaining the SNR of a trace goes as follows, first by using the next equation:

$$\mathrm{SN_{dB}} = 10\log_{10}\left(\frac{S_\mathrm{w}}{N_\mathrm{w}}\right), \tag{2.1}$$

where S_w is the signal window of the trace, and N_w the corresponding noise window. The signal window is defined such that it starts when the seismic event is recorded by the geophone. All data in this time window is squared and summed. The noise window is a window of the same duration as the signal window, but then measured before the event has taken place. The dataset used for this analysis can be downloaded from the KNMI Dataportal. This is an online database with seismic recordings up to 10 years back of the Groningen Gas Field area.

Since one of the objectives of this thesis is to look if also micro earthquake data can be used to find the moment tensor via machine learning techniques, the SNR analysis has been mostly done on these lower magnitude event data. To define what is meant by good, we can thus look at this SNR data. If the SNR is lower than 5, we could say the data will not be clear enough, i.e., there will be too much noise present in the traces for the network to distinguish between noise or signal. Looking at the Zeerijp area, we know that the region has a good coverage with a total of seventy-nine stations (2). This is for example visible in Figure 2.4. Here, the red line denotes the lower SNR limit that we can use to obtain a moment tensor from a trace. From the scaling of the horizontal axis one could already notice that there are a lot more lower magnitude events in this figure than higher magnitude events. Since this data is a downloadable data set (KNMI Dataportal) it only shows how important it is to also look at the question of training the neural network with lower magnitude data. The next thing we can observe is that the station coverage of the Groningen Gas field area is pretty good. From magnitudes bigger than 0.6 on, it shows that the SNR of the data is generally above the lower limit SNR line, at least for stations at close range. A more extensive research on the SNR has been done in (Verwijs 2021). To, quickly, summarize the outcomes of that research:

- Of the three component data, the E-component generally gives the largest SNR.
- Of the four station depth levels (0, 50, 150, 200m) the deepest level has data with the best SNR.
- Of the different bandpass filters tested, it became evident that a wider bandpass (1-10Hz) gives a better SNR for lower magnitude data, however, for the larger magnitude data, a bandpass filter of 1-4Hz is more suitable.
- In general all events with a magnitude equal to or larger than 0.79 should produce data with a suffiecient enough SNR.

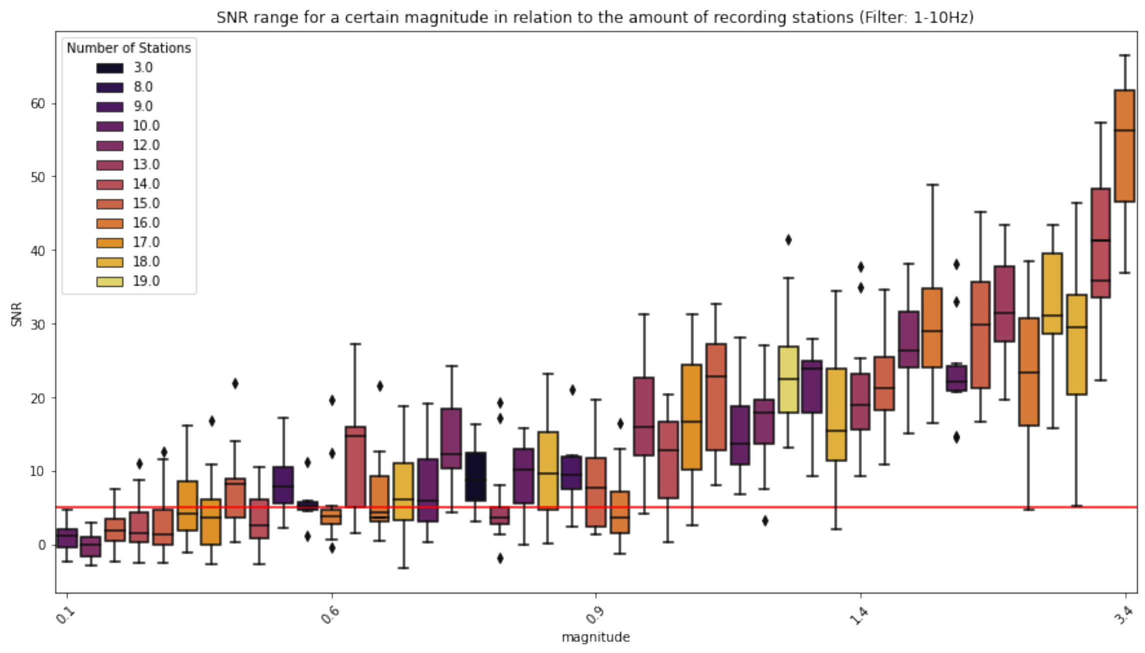


Figure 2.4: The SNR for data a specific magnitude. The color of the boxplot indicates the number of stations that have recorded the event and the red line at SNR=5 denotes the minimum SNR needed for an accurate result. As can be seen from this figure, is that even lower magnitude data has a pretty decent SNR. This is due to the good coverage in the area.

2.3 The Moment Tensor

If an earthquake occurs we can record the ground motion as a function of time due to this event. A natural earthquake is caused by movement along a fault. This can be strike-slip-, normal-, or reverse faulting. Before the event happens, the rocks surrounding the fault will experience a significant increase in shear stress. The stress will cause a build up of elastic strain energy. If the stored energy overcomes the friction on the already existing faults, an earthquake will occur. The two parts of rock slide along the fault and the stored energy will be released in the form of for example seismic waves. The amount of energy released is related to the magnitude of the event; the more energy, the higher the magnitude. The energy released cause the ground to shake, which in turn makes it able for us to record the earthquake. What can we obtain from these seismograms? Next to the magnitude, we could also obtain the location of the earthquake, if more seismograms from different stations in the area are combined.

It is also possible to obtain more physical properties of the earthquake, namely the faulting mechanism. The movement along a fault that causes the earthquake can for example be described by the following three parameters: the strike, dip, and rake. Each of these parameters explain a different property of the earthquake. Figure 2.5 shows how the strike describes the orientation of the fault with respect to North, the dip the angle of the slope of the fault, and the rake the displacement of the fault with respect to the foot wall. Yet, are these three parameters enough to describe the complete motion? Strike, dip, and rake can explain simple faulting mechanism pretty well, but in real life these systems look a bit more complex. It is therefore necessary that we use another system, for example the moment tensor. This representation depicts the seismic sources based on a system of forces (Dahm & Krüger 2014). The moment tensor is used to define the size, and motion of the source. If there is an area with high seismic activity, all moment tensors can help analyze the fault system. For example the analysis can be focused on the determination of type of fault. Moment tensor representations are mostly used for earthquakes, but can also be helpful for landslides, explosions and so on (Cox & Allen 2009) (Alvizuri & Tape 2018).

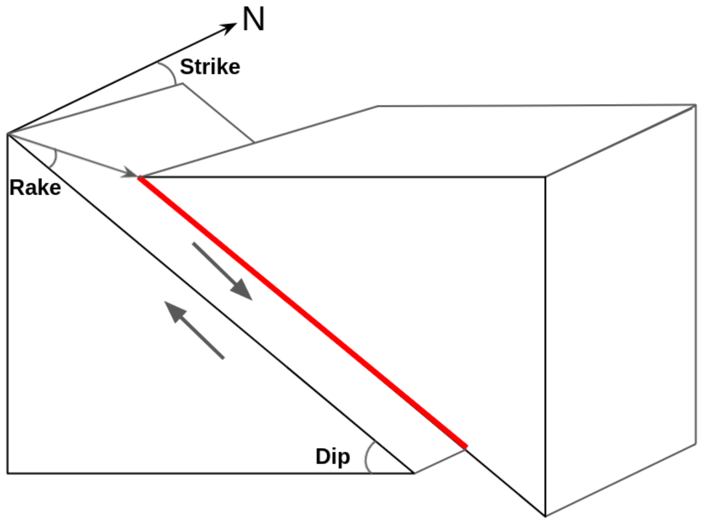


Figure 2.5: Strike, dip, and rake that explain the faulting.

A moment tensor representation can be derived using the waveforms recorded by the stations and via a mathematical description of the travel path of the seismic waves through the earth. The description of the travel path is called a Green's Function. Since in this thesis a precomputed data base of the Green's Functions for the Zeerijp area will be used, we will not go into detail on what exactly these functions are and how they work. If the reader is interested to learn more about Green's Functions, I would like to refer them to either (Wapenaar & Snieder 2007) for a gentler introduction, or (Wapenaar et al. 2010) for a more detailed explanation. Having obtained the computed Green's Functions, we can use forward modelling to create synthetic data, or we can obtain the moment tensor representation from recorded waveforms by, for example, using a full-waveform inversion. This can be done in the time domain, but also in the frequency domain. This does not mean that performing a full-waveform inversion to obtain the moment tensor representation is easy done. The first ideas were proposed some forty years ago (Tarantola 1984), and are still being improved today.

The moment tensor is a 3x3 matrix that is symmetric and thus has six independent components (Figure 2.6). The components on the diagonal describe volumetric changes, like dilatation or compression, and the off-diagonal components describe the rotational forces, like for example the shear stresses. These components combined make up the force couples that act at the source (Vavrycuk 2014). Therefore the moment tensor can represent any event, from a simple isotropic explosion to a very complicated seismic source, like the earthquakes in Groningen. To be able to find the type of source that is described by the obtained moment tensor, the latter needs to be decomposed. One possibility is to decompose the moment tensor into the isotropic- M_{ISO}, double-couple- (DC) M_{DC}, and compensated linear vector dipole (CLVD) M_{CLVD} components. The isotropic component and the DC component both represent a physical system, but this does not make the moment tensor mathematically correct yet. Therefore the CLVD component needs to be introduced (Knopoff & Randall 1970). It mostly accounts for the noise or assumptions being made, thus a small CLVD component can also be seen as a way to verify the accuracy of the model (Dahm & Krüger 2014).

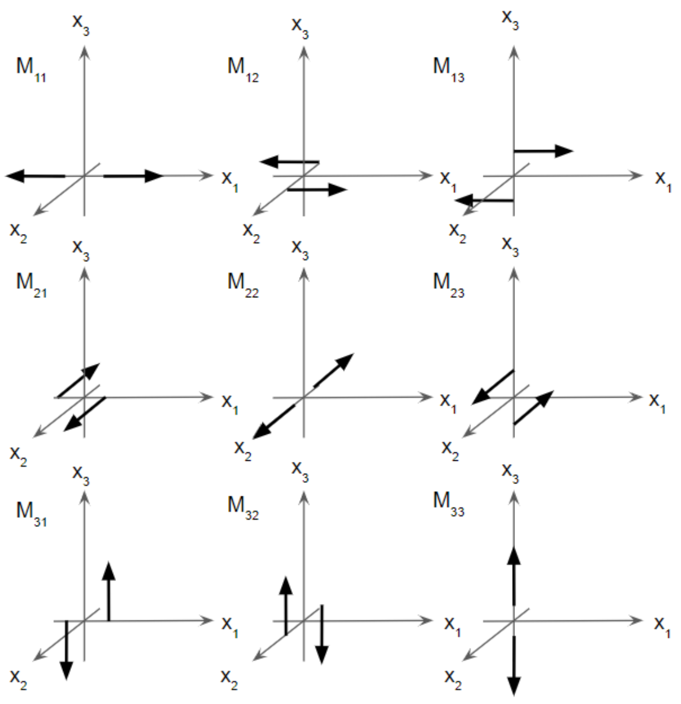


Figure 2.6: Moment tensor: visualization and decomposition into six independent components. On the diagonal the linear vector dipoles are shown, the other elements show force couples.

The isotropic component of the moment tensor represents an explosion or implosion. The DC represents shear faulting. To visualize the decomposition, a Hudson source type plot can be used, see Figure 2.7. In this Figure the CLVD, ISO and DC components are plotted into one system, which makes it easy to see the relative proportions of the components to each other. For example, in the Hudson plot, pure isotropic sources can be found at only two points in the diagram; the upper- and lower most points on the diagram. In other words, the deviatoric tensor (the CLVD and DC components) is zero, and this event is pure isotropic.

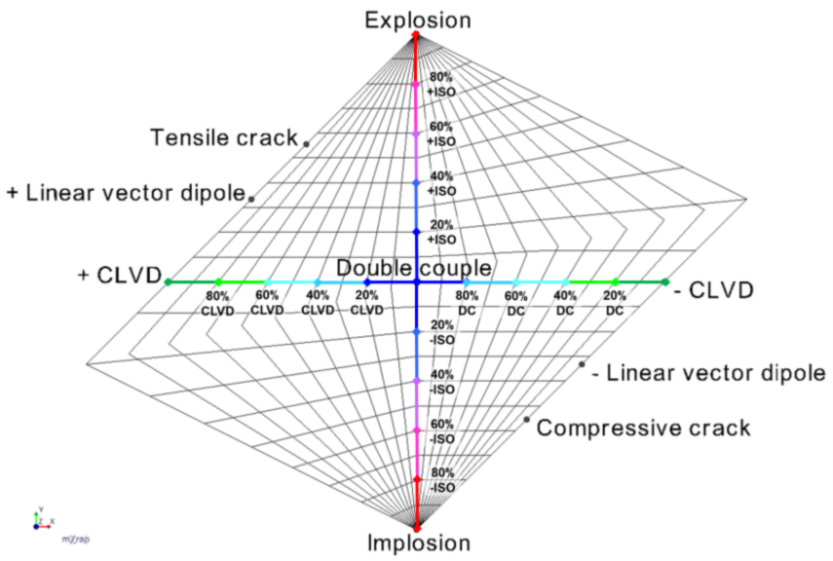
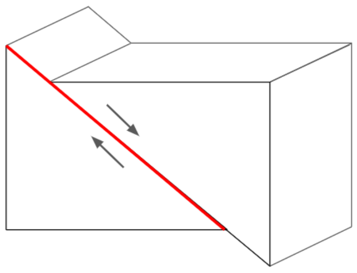


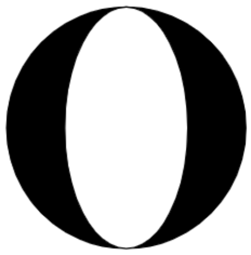
Figure 2.7: A Hudson plot visualizing the moment tensor decomposition (Tierney 2019)

From the solution to the moment tensor the focal mechanism can be derived. The focal mechanism describes the source in terms of strike, dip, and rake and can be visualized by 'beach balls'. Beach balls consist of white and black planes within a sphere that indicate the motion and type of fault. Here, the black and white planes represent the first arrival of the P-waves that are recorded. Simply said, if the source causes a push on the rock in a certain direction, there will be a compression. This compression causes the P-wave to arrive at the sensor with a positive sign. This compression is indicated by the black planes in the beach ball plot. If the rock undergoes dilatation, the first arrivals will be of negative sign. These P-waves are indicated by a white plane. In the area of the Groningen Gas Field normal faulting occurs (Figure 2.8a). The beach ball representing this type of fault is shown in Figure 2.8b. From the way the construction of a beach ball is described it can already be derived what factors influence the moment tensor inversion. If there is too much noise, the arrival of the waves will be less distinguishable, and if the different azimuths and take-off angles from the source are not recorded clearly, the reliability of the moment tensor will go down.

Retrieving the focal mechanisms of past earthquakes can give us more insight into the tectonic stress field of an area. Obtaining the moment tensor for several events can help understanding the induced seismicity in the the Zeerijp area better. Considering that the relation between the extraction of the gas and the seismicity itself is still not understood well enough, modelling a good fault map would immensely contribute to the monitoring and earthquake prediction of the Groningen Gas Field area. Not only is this of importance to the population in this area, who have suffered damages to private properties due to this increase in induced seismicity, but it is also of importance to the state and the industry, concerning the economical consequences.



(a) A Normal Fault



(b) Beach ball corresponding to a normal fault

Figure 2.8: The induced seismicity in the Groningen Gas Field is a result of normal faulting (Figure 2.8a), the moment tensor corresponding to this type of fault can be represented by the beach ball shown in Figure 2.8b

:

Chapter 3

Methods

In seismology, we can describe any physical earthquake source with moment tensors. We know how to obtain them from a seismogram, and how they can help us understand source mechanism. But is, for example, the full waveform inversion technique described in Chapter 2 the most optimal way of obtaining the moment tensor? Yes, it can be a pretty accurate method, but taking into account the computational power needed in order to obtain the focal mechanism is very high. Should there be some other way to be able to obtain these moment tensors with less computational power and/or time? One way is by using machine learning techniques, which have been around for quite some time now. Yes, for the estimation of moment tensors, it is quite a new topic. For example, (Binder 2018) shows that the moment tensors of micro seismic events can be detected from p-wave amplitudes. Another paper, by (Steinberg et al. 2021) shows that the moment tensor can be retrieved from just images of the amplitudes of the time traces that are categorized per event.
Machine learning is an overarching concept that encompasses different techniques. There are many subcategories under the term Machine learning, of which one is the (artificial) neural network. This neural network can learn from a set of data, after which it should be able to, for example, classify new data into its correct categories.

In this chapter we will first talk about the making of the synthetic data set. Having a representative data set is important for the network to be able to learn well. How to find the fine line between a data set that is too generic or too biased is discussed in the second section. Here, we will also discuss the different data sets used to find the answer to the question if it is possible to estimate the moment tensor components from time traces using a neural network. An overview of this section is visualized in Figure 3.1. Following this flowchart, the generation of the synthetic data set will be explained.
Having the data set ready, we dive more into the theory behind neural networks. What are they exactly and how do they work? Which parameters are important to consider when setting up such a network, and what triggers the network to learn or not to learn? Section 3.2 will elaborate on the feed forward network, also called the multilayer perceptron. As we will see is that this type of network will predict the moment tensor, however there will be no way of verifying the how certain the network is when making its predictions. This is where the mixutre density network comes into play (Section 3.2.3).
Having explained the conventional feed forward neural network, and the more advanced mixture density network, we can look at the differences between these two algorithms in terms of complexity and time intensity (Section 3.3). Can they complement each other and how have they been used before regarding moment tensor estimations? It is in this last section that we will explain the reason of using both the feed forward- and mixture density network on the basis of past research done on this subject.

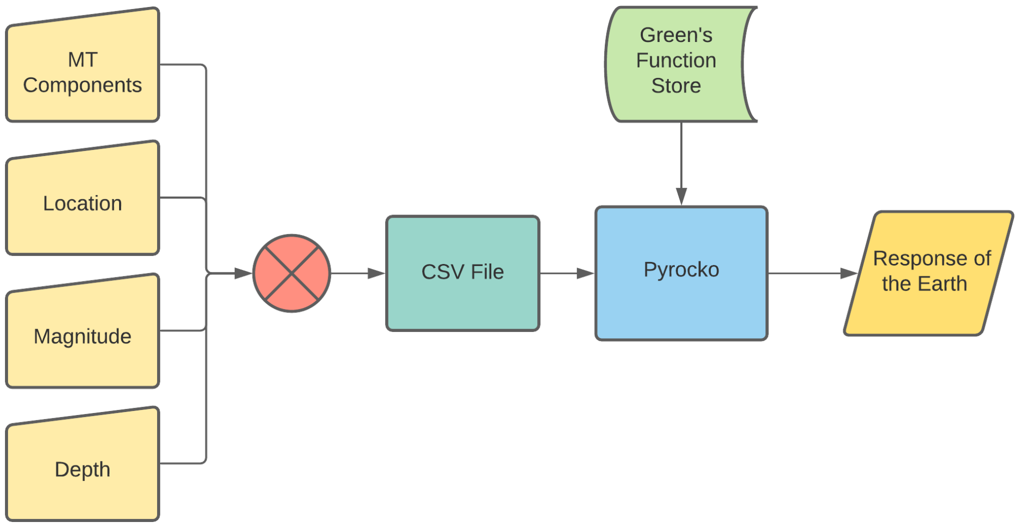


Figure 3.1: First, the moment tensor components (MT components), location, magnitude, and depth are defined for the event that is to be simulated. Then, all of this information, for each seperate event, is saved into a csv file. This csv file is then read by the pyrocko module, that in turn uses the precomputed Green's Functions to simulate the response of the earth based on the velocity model corresponding to the Green's Function Database.

3.1 Synthetic Data

Looking back at Figure 1.1 it can be seen that over the past five years alone over four hundred earthquakes have taken place in the area of the Groningen Gas Field. We could use this data to train our network, however most of these events were micro events, and thus do not have a very high SNR. Next to that, using real data would also not allow us to investigate the effect of difference factors like source location, type of model, type of faulting mechanism, and noise on the performance of our network. It is therefore that we propose to use synthetic data first to test the precision of the moment tensor prediction.
For the input of the network we are using the data from seismometers. These seismometers provide us a seismogram, which shows the ground motion at the specific location of the seismometer. To generate the synthetic data set, the python package Pyrocko (Developers n.d.) is used. One of the applications that is very useful is its online Green's function database that contains the velocity models plus corresponding Green's Function for the Groningen Gas Field area (`https://greens-mill.pyrocko.org/`). The two Green's Function databases used for this particular thesis are as follows:

1. '$N3Dmean\_buffer10km\_Q\_ER\_50m$' data set. This set contains the Green's Functions for the locally averaged 1D velocity model in the Groningen Gas field area. It has been modelled with the QSEIS backbend and allows earthquake sources with a depth range of 1000 to 4000 meters. The receiver depths can range from 50 to 200 meters, and the maximum distance from source to receiver is fifteen kilometers.

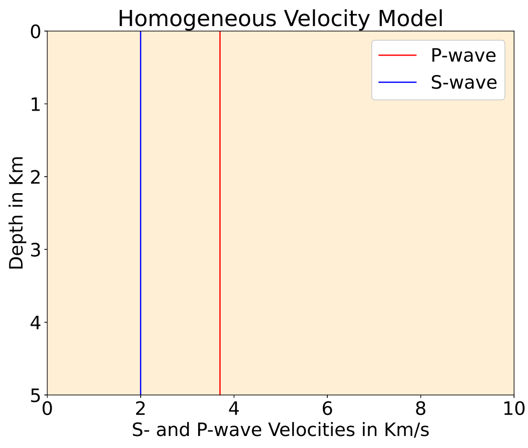
2. '*Gron_homo*' data set. This set contains a homogeneous velocity model specifically for the Groningen Gas Field area. The source and receiver specific settings are the same as for the locally averaged 1D model.

The velocity models of both data sets can be found in Figure 3.2. From these velocity models the Green's functions have already been computed, therefore there was not need to calculate them before generating the data set. Following the findings from (Kühn et al. 2020) and (Dost et al. 2020) the subsequent settings found in Table 3.1 were considered when generating the data set. See Figure 2.1 for the area in which the earthquakes are simulated (within the black box), the exact positions of the earthquakes following the existing faults in the area (the blue lines), and the position of the receivers (inverted yellow triangles).

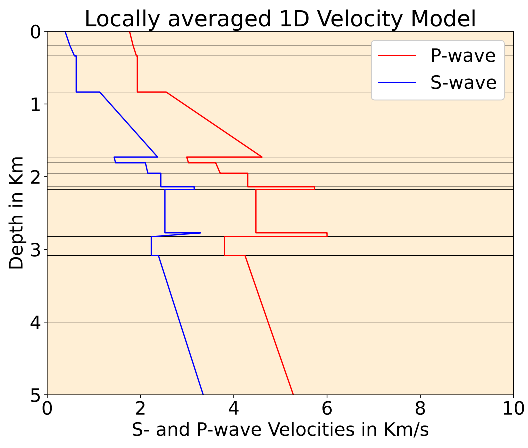
Parameter	Setting
Maximum distance from source to receiver	15 km
Start time	-0.5 seconds before start of event
End time	+9.5 seconds after start of event
Sample rate	25 Hz
Bandpass filter	1-4Hz
Source Depth	3 km
Source Magnitude	4
Recording receivers per event	13
Depth of receivers	200 m
Number of events	10.000

Table 3.1: Settings for the generation of the synthetic data set.

With the parameters from Table 3.1 two data sets can be created. One uses the homogeneous velocity model as shown in Figure 3.2a, and the other is based on the heterogeneous velocity model shown in Figure 3.2b. Table 3.1 might give rise to some questions on the similarity of the data set with regards to the real life events in the Zeerijp area. I would like to address some of the problems faced, and why certain simplifications had to be made to the data set.



(a) Homogeneous Model



(b) heterogeneous Model

Figure 3.2: The two velocity models used for the Green's Function Database used in this thesis that is specific to the Groningen Gas Field area. On the left the P- and S-wave velocity of the homogeneous model is shown over a depth of 10 km, and on the right the heterogeneous model containing the salt layers is shown.

Table 3.1 shows that one **magnitude** was used for the complete data set. The reasoning behind keeping this magnitude constant is due to simplification of the data set. This also means that the choice of bandpass filter has to be picked accordingly. As one might remember, earlier in this chapter (Section 2.2) it was stated that lower magnitude data has a higher SNR with a broader bandpass filter. However, since it was decided to fix the magnitude of every event to 4, the **bandpass filter** was changed to 1-4 Hz.
According to the testing (Kühn et al. 2020) did for their inversions a **receiver depth** of 100 or 150 meters should yield better resolution and a combination of different receiver depths even more so. However, also with the outcomes of the SNR analysis done in the earlier stage of this thesis, it was decided to only generate data for the 200m depth stations. The **depth of the sources** is also fixed to a single depth. This is mainly because in reality all faults in this area are situated around a depth level of three kilometer, but again, it also to simplify the problem in order to see if it will benefit the learning done by the neural network.

Looking at other papers on estimating the moment tensor using some form of machine learning, it is seen that most time windows used are not longer than four seconds (Steinberg et al. 2021), (Kuang et al. 2021). In this thesis a **time window** of ten seconds is used. The locally averaged 1D model causes some interesting responses (Figure 3.6). This makes it harder to determine where the important inf on the traces arrives and where the coda starts. To be on the safe side a time window of ten seconds was set. Another reason for this larger time window is the correlation found by (Käufl 2015), where it was seen that with a longer time window, the network proved to be able to extract more information on the focal mechanism from the signals corresponding to the subsurface of the Groningen Gas Field area.

Each source is generated from an unique, random moment tensor, location, depth, and magnitude. The individual moment tensor components range from -1 to 1, the location is drawn randomly from a csv file with all fault locations in the Zeerijp area. From the moment tensor the source response is calculated using the python package Pyrocko for all stations that are within the specified radius of five kilometers of the Zeerijp area (totalling to an amount of 13 receivers), see Figure 3.3. For each earthquake simulated the source location and six independent moment tensor components are saved. These will later be used to help train the neural network. An example of one such an event is shown in Figure 3.4. Here, one can clearly see how the response arrives delayed as a function of distance from the source.

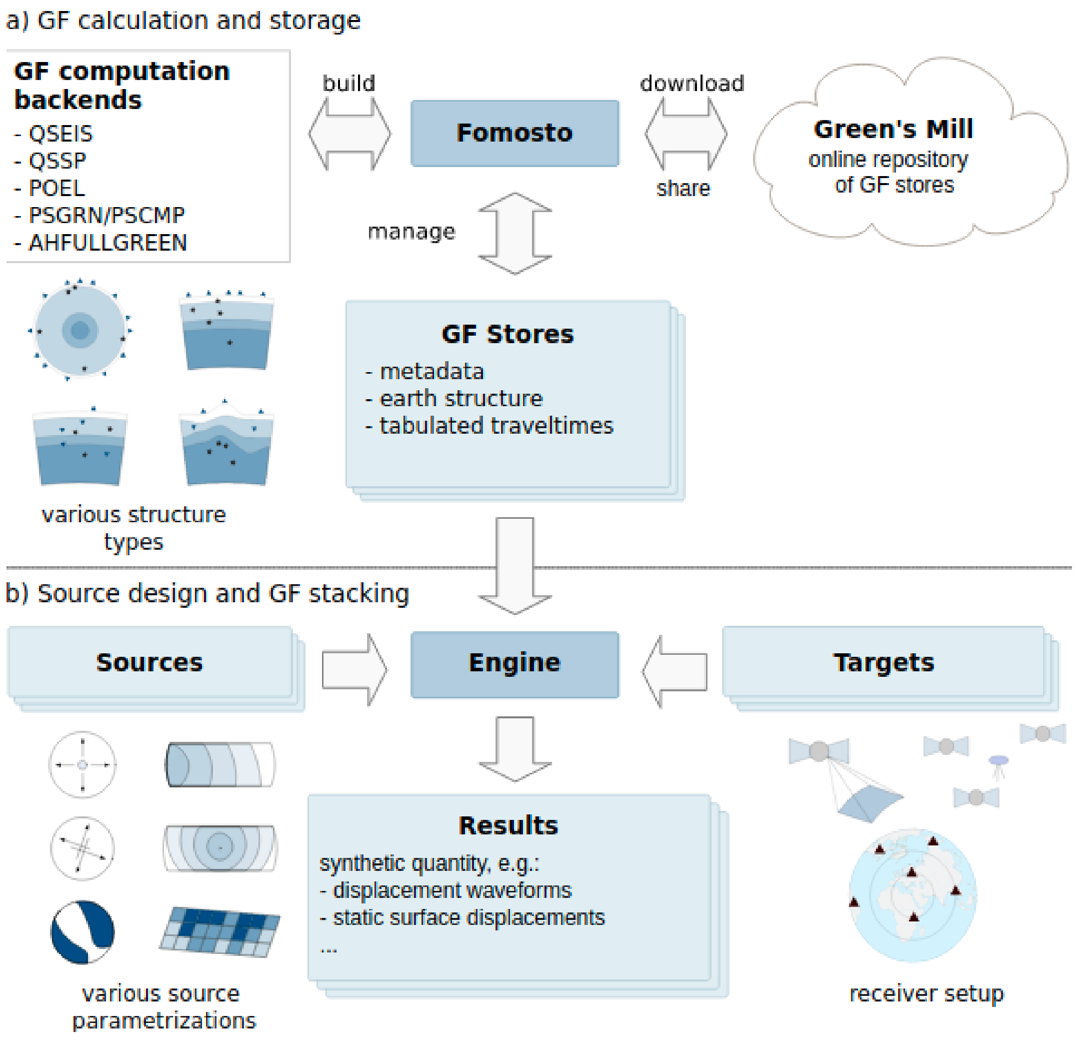
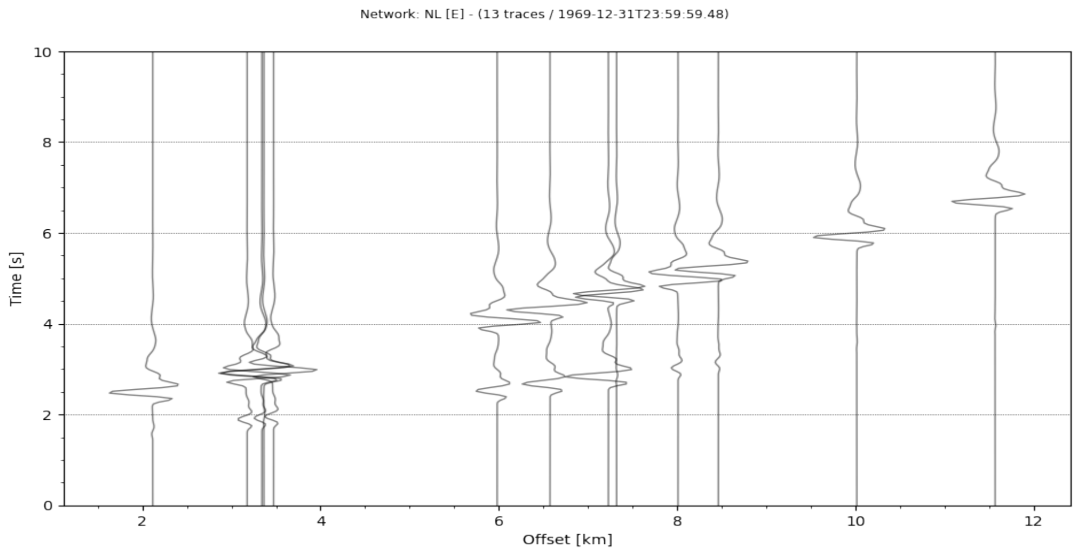
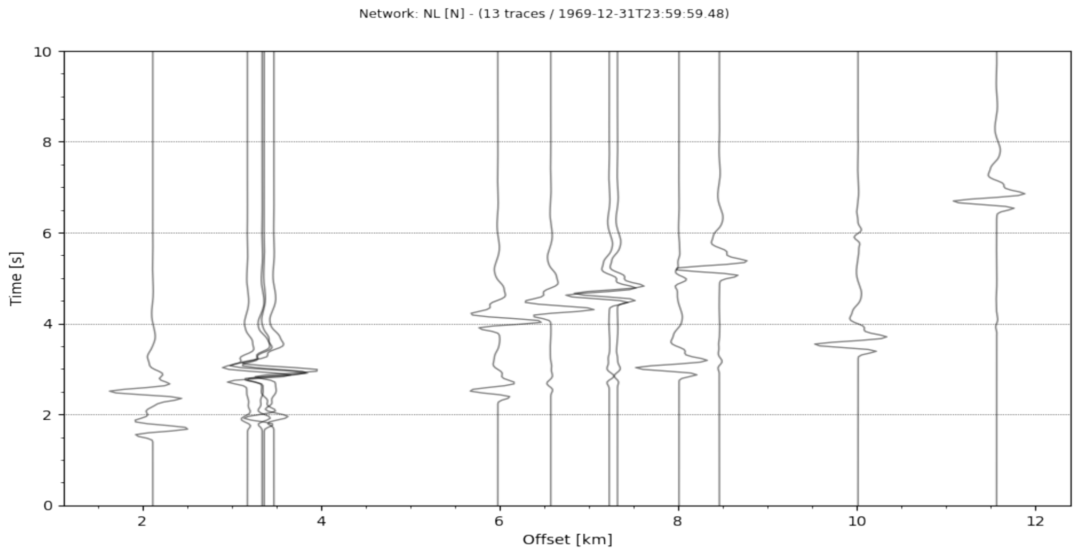


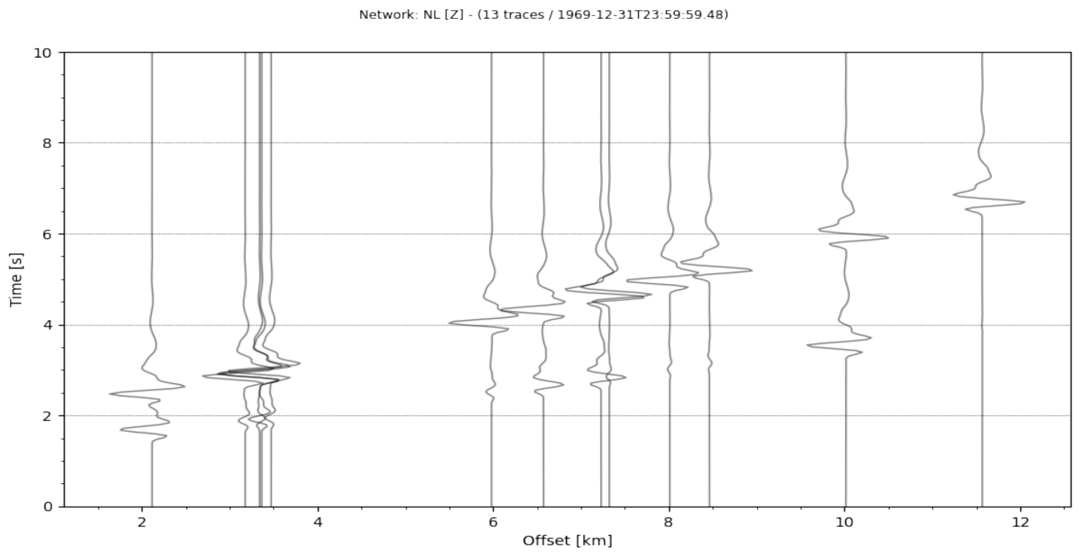
Figure 3.3: Flowchart forward modelling with Pyrocko, source: (Developers n.d.).



(a) Channel E



(b) Channel N

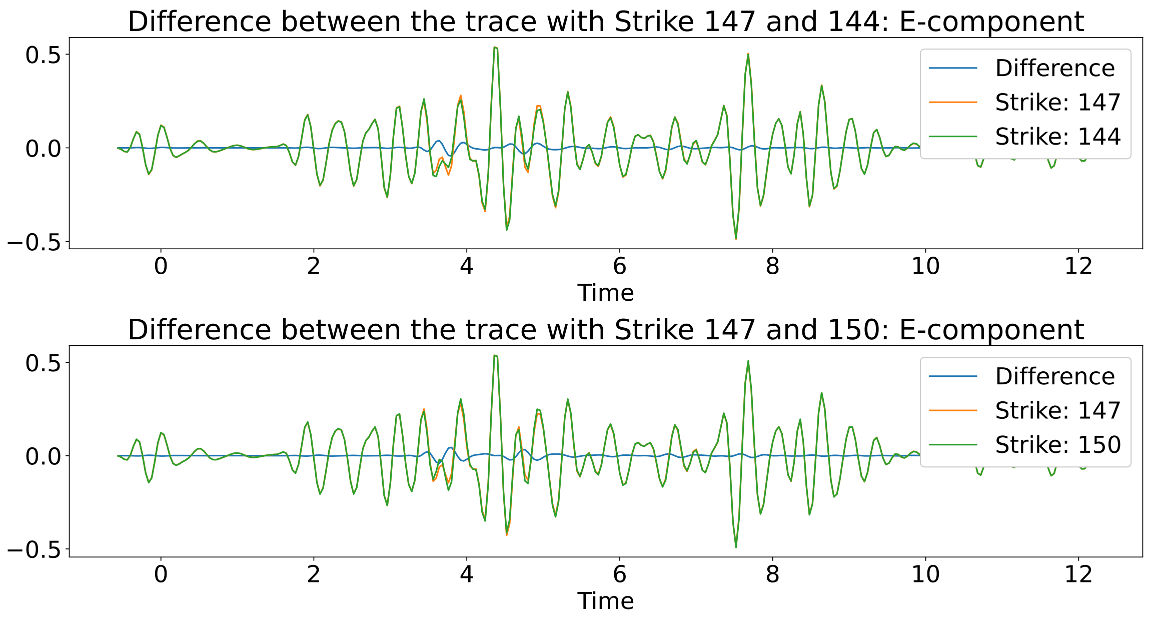


(c) Channel Z

Figure 3.4: Response of synthetic event 1 for all receivers within the predefined Zeerijp area. From top to bottom, the E, N, and Z channel responses filtered with a bandpass filter of 1-4Hz. Note that with increasing distance, the first arrival also delays.

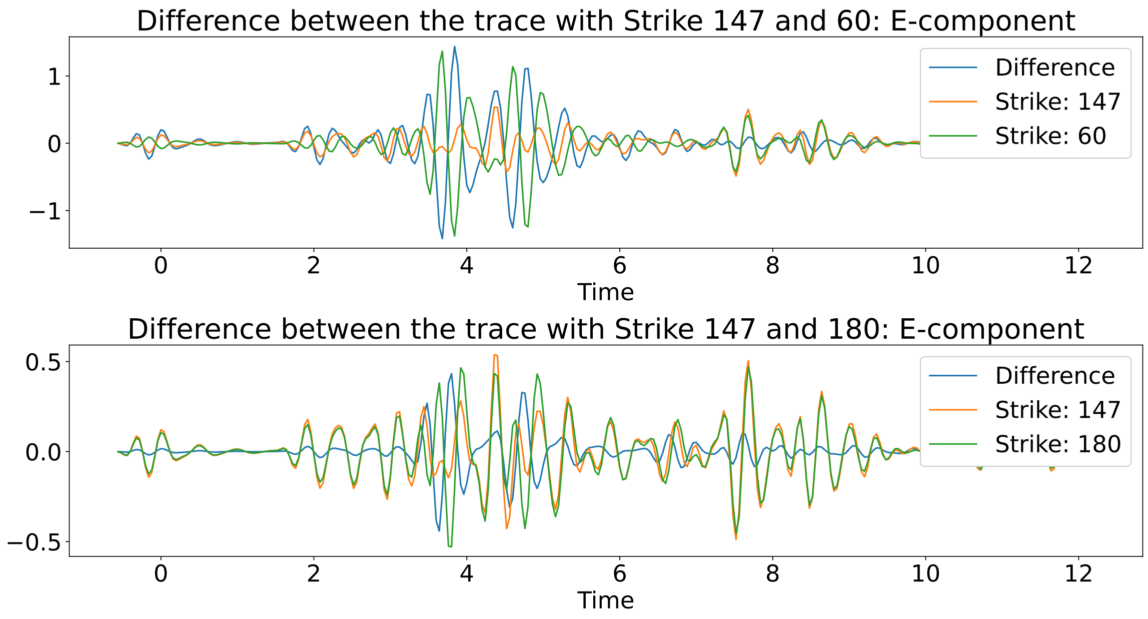
It is important to have a representative synthetic data set. This means that the representative data set defines data that is not too biased, but also not too different keeping in mind the earthquakes that are taking place in the Zeerijp area. For example, a data set with around ten thousand synthetic earthquakes could be a good start according to (Binder 2018), however, if this data set only represents five different source mechanism, whereas we want to estimate a much wider range of faults, it could not be called a representative data set. (Dost et al. 2020) found already thirteen different moment tensors for the area of the Groningen Gas Field. Biasing our complete data set towards these thirteen events could be a strategy, but it would also mean that we do not consider other types of faulting mechanisms anymore. It is therefore key to find a good compromise. How can we find this fine line? It might be best to first look at how a change in the moment tensor influences our traces.

The traces in Figure 3.5 were made using a 1D averaged velocity model from the 3D velocity model by (Romijn 2017*b*) (Figure 3.2b). This velocity model is based on the subsurface found in the Groningen Gas Field area. The figure shows us an E-component trace from a station that is 4.2 kilometers away from the source, and the source mechanism having a strike of 147, dip of 64, and a rake of -92 (yellow trace). If we make just a slight change to the source type, the trace does not change that much (green trace). For comparison, the blue trace shows the difference between the yellow and green traces. Figure 3.5b on the other hand shows how big the change in strike resulting in a visible difference between two traces.



(a) Small change in strike

Figure 3.5: Figure 3.5a: Two traces, yellow and green, from slightly different sources of which the difference between the two is plotted (blue line). The upper figure shows a reduction in strike from 147 (yellow trace) to 144 (green trace). This change does not yield that much difference in the corresponding traces. The difference between the two traces is visualized with the blue line. The same can be said for an increase in strike (lower figure). Figure 3.5b: Two traces, yellow and green, from different sources of which the difference between the two traces is plotted (blue line). The upper figure shows a reduction in strike from 147 (yellow trace) to 60 (green trace). This change does yield difference in the corresponding traces. This difference between the two traces is visualized with the blue line. increasing the strike to 180 also shows a change in trace, however it is less obvious as with the previous change.



(b) Large change in strike

Figure 3.5: Figure 3.5a: Two traces, yellow and green, from slightly different sources of which the difference between the two is plotted (blue line). The upper figure shows a reduction in strike from 147 (yellow trace) to 144 (green trace). This change does not yield that much difference in the corresponding traces. The difference between the two traces is visualized with the blue line. The same can be said for an increase in strike (lower figure). Figure 3.5b: Two traces, yellow and green, from different sources of which the difference between the two traces is plotted (blue line). The upper figure shows a reduction in strike from 147 (yellow trace) to 60 (green trace). This change does yield difference in the corresponding traces. This difference between the two traces is visualized with the blue line. increasing the strike to 180 also shows a change in trace, however it is less obvious as with the previous change.

This sort of analysis can be done for all components and a short overview of the outcomes can be seen in Table 3.2. The Table contains three main categories; velocity models, source specific changes, and receiver specific changes. In the first category, velocity models, it is seen that three velocity models are compared to each other. These three models are the models used (Kühn et al. 2020) in their paper to analyse which velocity model will work best for the inversion of the data and represent the area in Groningen the best. They can be found in the Appendix (Figure A.1). The first type of velocity models are the ones derived from the available 3D model for this specific area (Romijn 2017*a*), which in total accounts to seventeen 1D velocity models. The second type of velocity model is the model from the KNMI. This model has a constantly increasing P-wave velocity and describes only big layers, therefore merging smaller layers together. The third model follows the local velocity model better, however it contains two thin anhydrite layers that cause high velocity perturbations (Kraaijpoel & Dost 2013).

A nice example of the difference between a homogeneous velocity model, and the velocity model that will be used for this thesis (the locally averaged 1D velocity model, see Figure 3.2b), is shown in Figure 3.6. We can see three different, simple source mechanism displayed. The homogeneous model (the green traces) show a very ’clean’ trace, in comparison to the response for the locally averaged 1D model (blue traces). This is what we would expect. A homogeneous model does not contain contrasting impedances, since there are no layers defined in the model. The locally averaged model, however, has the Zechstein formation modelled into it, which we know contains salt. This should make the response a lot more difficult to interpret, something we can recognize by looking at the difference between the middle and lower row of traces in Figure 3.2.

For the second category, the source specific parameters, it is seen that especially the depth of the source causes notable changes to the traces. The change in depth, however, is not a factor that is investigated in this particular thesis in relation to the moment tensor estimation. As can be seen in Figure 2.3a the reservoir rock is mainly situated around the 3000 meters depth mark throughout the whole cross-section.
One could ask why the influence of the magnitude moment of the event has not been investigated. It is simply

because a change in magnitude moment will only result into a change in amplitude.

The receiver specific parameters show that whatever setting is changed, there will always be a notable change in the response. This is pretty logical, since the farther away or closer to the earthquake the station will be placed, the fainter or stronger the signal will be. Here, I have only looked at distance of the stations to the source location (instead of the longitude and latitude), since, to keep the data set as realistic, only the real locations of the stations in the Zeerijp area will be used during the generation of the synthetic data set. For a more extensive analysis of parameters and the changes they cause, see Appendix A. Trace Analysis.

Parameter	**N-component**	**E-Component**	**Z-Component**
Velocity model (3 models)	$+$	$+$	$+$
Source Specific			
Strike	$\sim$	$\sim$	$-$
Dip	$\sim$	$+$	$+$
Rake	$\sim$	$\sim$	$-$
Depth*	$+$	$++$	$+$
Longitude	$+$	$\sim$	$-$
Latitude	$+$	$\sim$	$-$
Receiver Specific			
Depth	$+$	$+$	$+$
Distance	$+$	$+$	$+$

Table 3.2: Three categories (Velocity model, source specific-, and receiver specific settings) shows how much a response is changed by a small alteration of one parameter. Here, a $-$ denotes no change visible, a $\sim$ a little bit of change, and a $+$ or $++$ a notable change or significant change respectively. A more extensive analysis can be found in Appendix A.

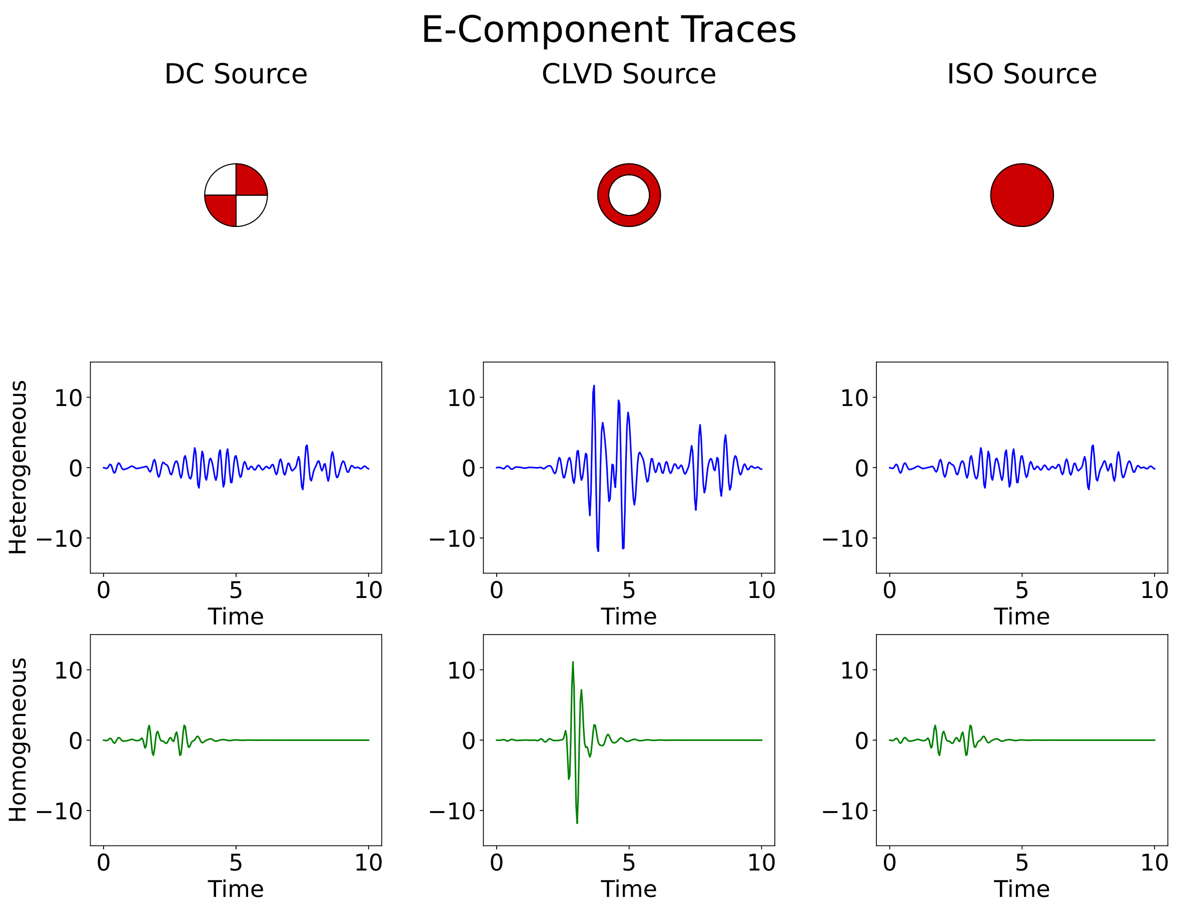


Figure 3.6: Three different, simple source mechanism and two different velocity models. Next to showing how the source mechanism influences a seismic trace, it also shows how the complexity of a velocity model affects the response. In this case we can see a clear increase in complexity for the locally averaged model (blue traces) in comparison to the response of the homogeneous velocity model (green traces).

From our previous analysis differentiation between two responses of almost identical source mechanism proved to be hard. Training with a bigger data set will make the possibility larger that traces will look more like each other, since moment tensor components only vary in the range -1 to 1. As is also the case when using full waveform inversion to obtain the moment tensor, we need a certain azimuthal coverage that will ensure there is enough information on each event, making sure the network can differentiate between different fault mechanisms. This can be achieved by letting the network analyze multiple station data at once.

3.2 Artificial Neural Networks

Around the fifties of the last century the first ideas for a neural network were proposed. This concept was taken by other scientist and elaborated on, however, it did not take long before the idea of a neural net was forgotten again. It turned out that they did not work as well as was hoped for. Roughly one decade ago neural networks have made their return in science, and more and more research using these algorithms is performed nowadays. The term neural network can mean a variety of types of networks. In this thesis we will only look at the type of feed forward neural networks, or a multilayer perceptron. First, we will start of with a gentle introduction to the theory of a feed forward neural network, where, among other things, we will lightly touch the subject of back propagation as well. After this introduction, an explanation of the different hyperparameters and strategies that can be set and employed when training with such an algorithm are given.

3.2.1 Theory

The easiest way of understanding how a neural network learns is by seeing it as a kid who is learning how to multiply. Through looking at a lot of the basic tables, it learns which multiplication leads to which answer, in the end, making him or her able to do all multiplications she or he an come up with, without supervision of, for example, a peer or adult. This is also where the neural in the name comes from. The whole model is loosely based on the concept of neurons in a human brain that, wired together, can learn multiple things. Inside a neural network multiple layers can exist, where each layer contains a certain amount of neurons. These layers within the network is what we call hidden layers. Each of the neurons learn specific mathematical functions that, when used all together, should be able to give us the desired output. We know how the model updates itself (more about this later), however, we do not know what the network learns exactly. As defined in Section 3.1 we chose a specific time window because we think the network will learn most from these first arrivals. This is a form of feature selection and can help someone training a network obtain a better accuracy. However, it is still hard to know what precisely triggers the network to learn from the data what we want it to learn. If training with a convolutional neural network (CNN), it would be a bit easier to see what the network is actually learning from the data we feed it. A CNN is a type of network that learns filters in the hidden layers, which are applied to images fed to the input layer to predict the output value. In the simple case of a CNN that estimates if a picture shows a cat or dog, we could visualize what kind of filter each hidden layer learns. An example of such a filter is a filter that picks out pointed or rounded ears, thus, first, giving a clear idea of what the network learns, but, secondly, also providing a good way of distinguishing between the two classes, namely a cat or a dog.

Finding out what exactly a feed forward neural network learns is more difficult to visualize, since it learns a bunch of linear functions. The parameters inside this function that are learned is what we call the weights and biases of, for example, a neuron. How this works is as follows: say we want a trained network that can give you the location of the source when you use a single time trace from a certain receiver as the input (visualized in Figure 3.7). Then, if we start feeding it random time traces, the network will try to learn the locations from it. When it is finished training, we can feed it a traces that the algorithm has not seen before. If we record a trace, it will have a sample rate, for example 25Hz. This means that we measure the earth's response each 0.04 seconds. Every time shot that we record can be seen as one input neuron of the neural network, meaning that if we have a traces with one hundred amplitudes (that shown together will form a wavelet), we need to construct a network that has a hundred neurons as input. The neurons of the input layer are all connected to each neuron of the first hidden layer, where the different neurons will all multiply the output from the input layer with a learned parameter. This parameter is what we call the weight of the neuron. Intuitively we think the network will learn the location of the earthquake partly by how long it takes for the first signals to arrive at the station. So, for example, if the time trace we are using is ten seconds long, and the signal only starts after the first three seconds have passed, we hope that the network will pick up how long it takes for the first p-waves to arrive and relate this to the distance to the source.

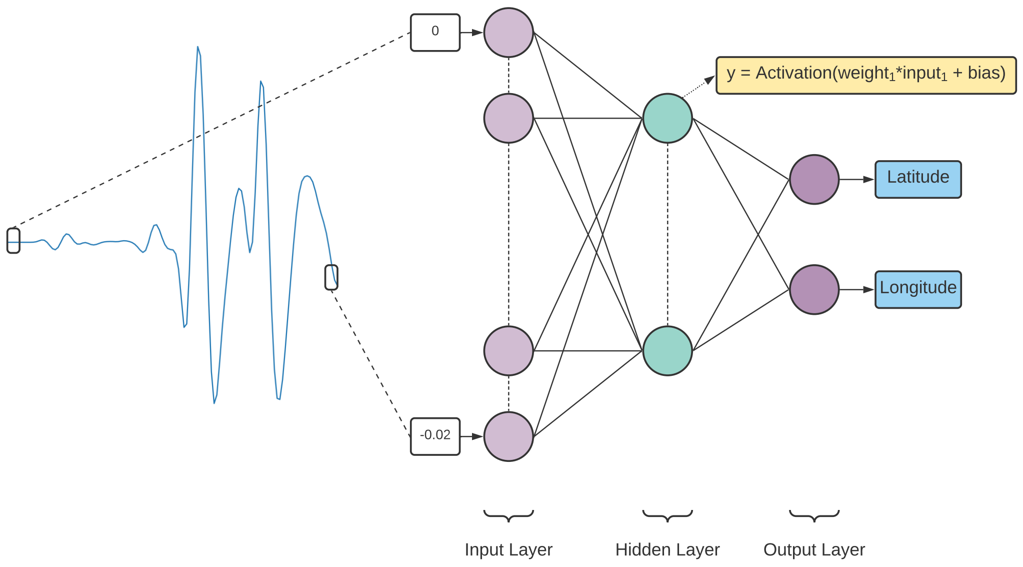


Figure 3.7: A time trace is recorded with a certain sampling rate. For each sample recorded, one input neuron to the neural network is created. The input layer forwards the data to the hidden layer, where it is used in a linear function, of which the output is transformed and prepared for the output layer. The output layer contains for each output an output neuron, in this case giving us the predicted values for the latitude and longitude.

But what if the first arrivals are not strong enough to be picked up by the network or are so weak that they will not survive the chain of multiplications and neurons it will have to go through, becoming weaker with each pass? It will mean that important signal will be lost. It is therefore that we add a certain bias, to be able to overcome a certain threshold before one neuron can activate another neuron, see Figure 3.8.
After adding the bias we have the 'final' product of our input with the neurons in the first hidden layers. As said before, each mathematical function inside the neurons can predict only a linear function, since the weights and biases of each neuron are linear. An activation function introduces the non-linearity to the network and enables it to learn almost any function there is. It takes an input, and via a transformation (a non-linear function) it gives the output. Examples of such an activation function can be found in Figure 3.11.

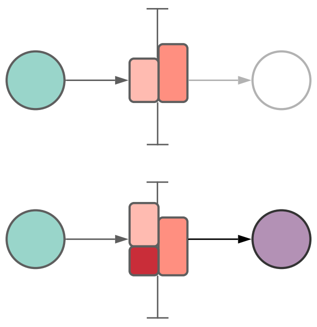


Figure 3.8: In the upper figure we see a neuron that produces an output that does not meet a certain threshold to activate the neuron in the next layer, however, this link between the two neurons might be an important link for the network later on. By adding a bias to the neuron, we can see, in the lower figure, that now the output of the neuron does overcome the threshold and consequently activates the neuron in the next layer.

Knowing how the architecture of a simple feed forward neural network can look like, we might ask ourselves how the networks learns. As said before, we do not know what exactly the network picks up, but we do know how it updates itself. The parameters inside the network, i.e., the weights and biases, are updated using back propagation. First, how does the network know if it predicted something that makes sense? For each training sample we feed to the network, we have a corresponding label that tells us what the network's output should be. If we go back to the example used before, where we were trying the network to learn the location of a source from a single trace, the first ever sample we feed it will probably give a very bad prediction. Comparing this prediction of the location to the actual location, we can calculate the loss between the two. The definition of the actual loss function, but also its shape varies on what type of problem you are dealing with. In the end we want to minimize this error between the predicted and true values as much as possible, because it will give us an indication that the network is performing well. It is therefore vital to tweak the right parameters to optimize the learning.
In the case of a single hidden layer network, the algorithm only needs to look at how to change the mathematical formulas in this one layer, however, when dealing with multiple hidden layers, this process becomes increasingly more complex. This updating of each single neuron, that is then related to another neuron is what we like to call back propagation. Figure 3.9 gives a feel of how the back propagation method works. When the network processes a sample, like in Figure 3.7, it will at the end check how much the predicted sample differs from the true value. Then, all links between neurons, and the neuron specific parameters themselves will be updated accordingly. This is done via the gradient. The gradient of the loss function with respect to the biases en weights of all individual neurons in a network are computed, which in turn tells the network how these weights and biases of each single neuron should be adjusted to reduce the error. The parameters can be updated a little, a lot, positively, or negatively. If, wishing to know more about the theory behind back propagation, I would kindly refer the reader to (Bishop 2006), which is a great book that explains all there is about machine learning in more detail.

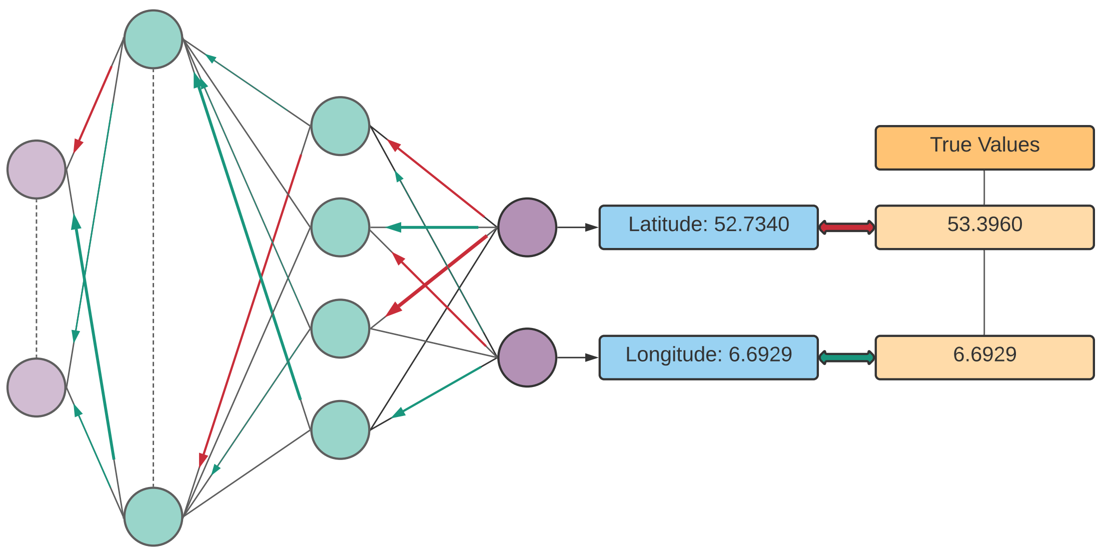


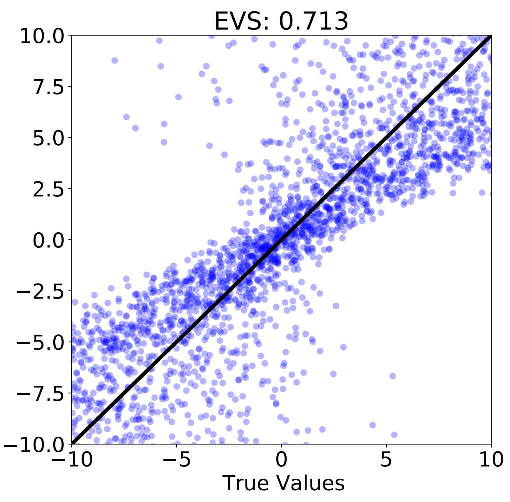
Figure 3.9: Once a sample has propagated through and been transformed by the network, arriving at the last layer the estimated output will be assessed by computing the error between the true and predicted value. If there is an error, the network will update itself by making changes to parameters in the algorithm via a backwards motion. This means that the network will adjust the parameters in a motion form right to left in this figure. Here, the back propagation can be a small or big change, positively or negatively.

3.2.1.1 Explained Variance Score

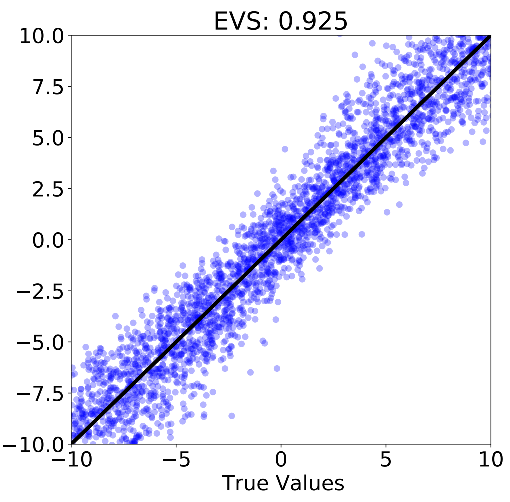
How do we verify the accuracy of the algorithm? The term accuracy is mostly used when dealing with a classification problem. In this thesis, facing a regression problem, the validation of the performance of the network is a little bit different. Usually a way of verifying if the network is trained well is by plotting the predicted values of the test set against the target values that correspond to the input of the predicted values (Figure 3.10) (EVS). This way we can visually check the performance of the network. Despite this visualization it can still be pretty difficult to assess the accuracy of the network. Up to some degree it is certainly possible to say if a network predict very poor, medium, or very good, but what about everything in between? A solution to this problem is by (among other techniques as well) using the **explained variance score**. This score gives scores in the range of minus infinity to one, with the latter being most desirable, since it indicates that the input is perfectly predicted by the algorithm. The formula to calculate this score is

$$explained\_variance = 1 - \frac{Var\{y - \hat{y}\}}{Var\{y\}}, \tag{3.1}$$

with y being the true values of the input, $\hat{y}$ the predicted values, and Var the variance. This score explains how much of the variance is accounted for in the prediction of the neural network when the input are the previously unseen samples. For classification problems the measure of the performance is the accuracy of the model. We can either have one class or another, and how good the network is at classifying each sample results into the accuracy of the model. This is a simple score that feels very intuitive; given class A, there is a ninety percent change of the network guessing the class right. A regression problem cannot make use of this type of accuracy, since we want to predict continuous values. So, even if we reach an explained variance score of almost 0.9, i.e., a very 'good' score, this is not synonymous to the accuracy of the network, meaning we cannot say that the prediction of this network should be taken with a ten percent error margin. An example of this is shown in Figure 3.10. Here, we see two models, both with a different explained variance score. Looking at the model with the EVS of 0.925, and how the true versus predicted values cluster around the desired, black linear line, we could say, yes, the error margin of our predictions could very well be around eight percent. But, using this same concept on the model with an EVS of 0.713 we see this will not work, since for some parts of the model (for example in the range of true values -10 to 0) the error margin is not symmetric. This uncertainty can be investigated more by the use of mixture density networks, which we will talk about in Section 3.2.3.



(a)



(b)

Figure 3.10: Verifying the performance of a neural network for a regression problem can be done two ways. The first is by visualizing the target versus the predicted values, the second one by using the explained variance score. In both the left and right figure the true versus predicted values are plotted for a toy problem. We can make use of the explained variance score, in this plot shown as 'EVS'. A score of 1 is the highest possible and at the same time the most desirable result, meaning that the network can explain the variation in the data perfectly, and the further off from this value (lower than one, or even negative) indicates a poor performing network.

3.2.2 Hyperparameters and Strategies

We now have fed the first sample to our network. It did not predict that well, one of the locations was off (Figure 3.9), so what can we change now in order to make it predict the location better with the next sample? There are multiple hyperparameters and strategies that can be changed, and these changes also indirectly influence other settings in the network. Next to the network architecture, it is vital to pre-process the data set. For example, scaling the input data can work as some form as regularization, thus making it possible for the network to learn better.

We will distinguish between two sorts of parameters that we can change while improving the neural net; the hyperparameters and the strategies. With hyperparameters we mean all parameters that are used to control the learning process. Strictly talking, a parameter, in the case of a neural network, is a value that is learned during training, for example the bias of a neuron. Strategy in this case means every method in which we can constrain the model from overfitting. This can be through regularization, scaling of the input and target data, or by choosing a different optimizer. It is important to understand that the strategies are not perse single, numerical values (like hyperparameters), but are more some sort of technique that help us fine tune the network.

The hyperparameters are parameters that influence the learning process. Under this category we define the number of hidden layers and neurons in a network, but also the activation function used, the learning rate, momentum of the learning, the amount of epochs for which the network is trained, or the batch size. Within this section, all strategies and hyperparameters changed during this thesis to optimize the learning process of the network are elaborated on. Table 3.3 at the end of the section provides an overview of all hyperparameters and strategies discussed.

3.2.2.1 Activation Function

Take for example the **activation function**, which has already been mentioned in the previous section, it is a function that helps training the network by transforming the output of a certain node to a new, bounded range. The activation functions are mostly non-linear, thus capturing the non-linearity of our model. This is very important, since our problem is non-linear as well. There are different type of activation functions, each of them corresponding to a certain problem. Take for example a sigmoid activation function (Figure 3.11a). This type of activation function works very well for classification problems, since a majority of the input will be mapped to either a value of 1 or 0. The equation for the sigmoid activation function is as follows

$$\sigma(x) = \frac{1}{1+e^{-x}}. \tag{3.2}$$

The next activation function is the Rectified Linear Unit (ReLU)function (Figure 3.11b). This function is mostly used in deep learning, and computes a lot faster than for example the sigmoid activation fuction. The ReLU function is two linear functions combined to one non-linear function,

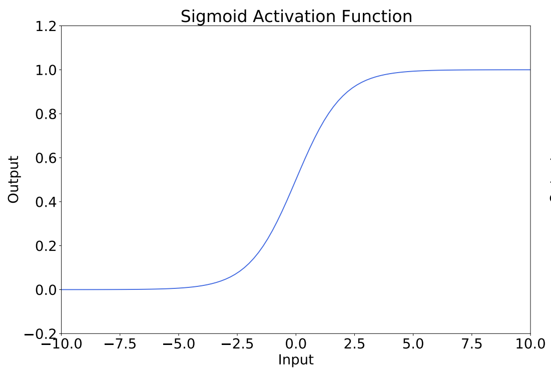
$$\begin{cases} 0 & \text{if x} \leq 0, \\ x & \text{if x} \geq 0. \end{cases} \tag{3.3}$$

On of the problems when using the ReLU is the effect of the dying ReLU problem. This might happen when the learning rate is too big. This causes neurons to only output negative values, thus causing the ReLU activation function to transform all these outputs to zeros. One way of countering this problem is by selecting a SELU activation function, instead of the ReLU function. Here, SELU stands for Scaled Exponential Linear Unit, and it is a variation on the ReLU function first proposed by (Klambauer et al. 2017), who showed that using a SELU function will self-normalize the network. This means that the vanishing gradient problem will disappear. The SELU function is shown in Figure 3.11c and is defined as follows, with *scale* and α pre-defined constants,

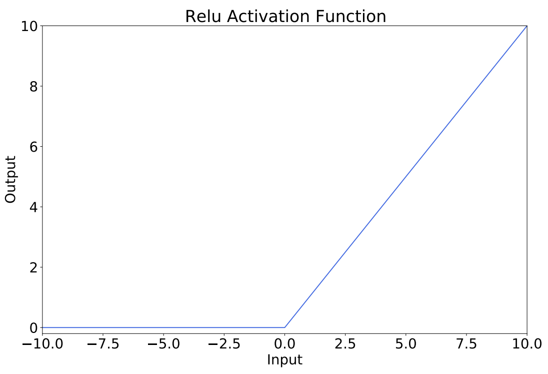
$$\begin{cases} scale * x & \text{if x} > 0, \\ scale * alpha * (e^x - 1) & \text{if x} \geq 0. \end{cases} \tag{3.4}$$

Very similar to the SELU is the Exponential Linear Unit (ELU). This activation function solves the same problems as the SELU function, however, the equation does not contain the *scale* parameter,

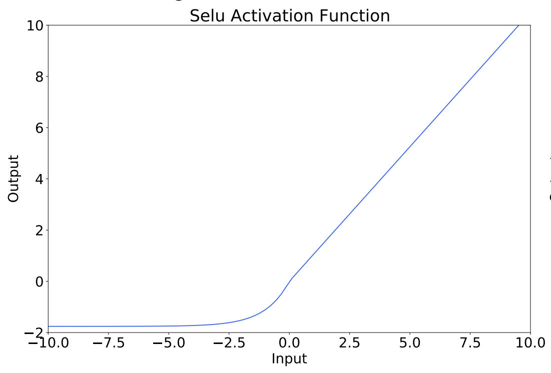
$$\begin{cases} x & \text{if x} > 0, \\ alpha * (e^x - 1) & \text{if x} \geq 0. \end{cases} \tag{3.5}$$



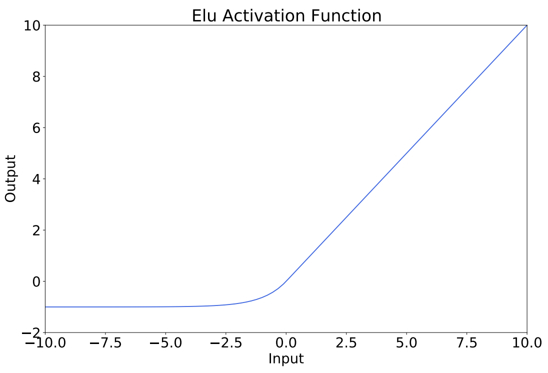
(a) Sigmoid Activation Function



(b) ReLU Activation Function



(c) SELU Activation Function

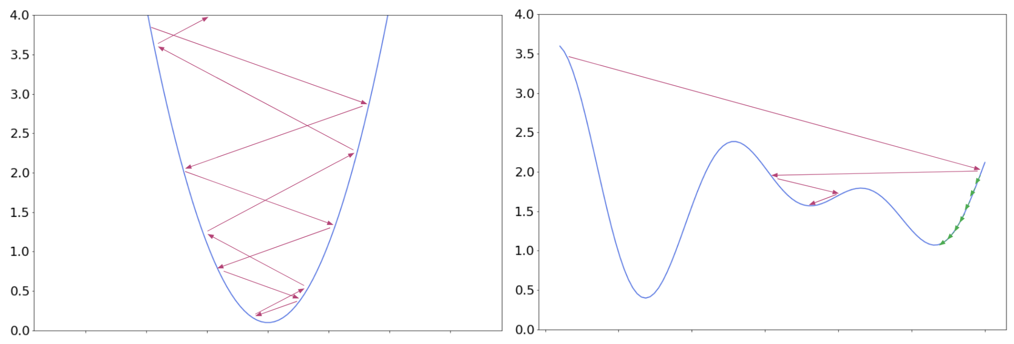


(d) ELU Activation Function

Figure 3.11: Four different activation functions, please note the changing output axis per activation function. The sigmoid activation is used for classification problems, the ReLU activation is mostly used for regression problems, and the SELU and ELU activations are found to work best for most problems in this thesis.

3.2.2.2 Learning Rate

The next hyperparameter that is important when tuning the network is the **learning rate**, and the related momentum. In the previous section the concept of back propagation was explained. After the error between the true and predicted output is calculated the learning rate controls how much should be changed to the model, before the process starts over. Setting the right learning rate can be tricky, since a lot of times we are dealing with global or local minima. The loss function that we use is in the network to minimize the error between the true and predicted values tries to find a minimum, however, it is not always the case that there is only one minimum possible. We could for example set the learning rate to a really low value. As a consequence we will find the global minimum, however, the time it will take to get there can take very long. On the other hand, by setting a higher learning rate, and thus effectively shortening the time of training, we could overshoot the global minimum and find only a local minimum, or worse, not find a minimum at all.
This concept is visualized in Figure 3.12. In both the simple and more complex case, the red arrows show how choosing a learning rate that is too big can cause the network to either not converge, or converge to a local minimum. The latter can also happen when picking a learning rate that is too small, i.e., the green arrows show how a too small learning rate will also converge to a local minimum, instead of the global minimum.



(a) One Minimum (b) Multiple Minima

Figure 3.12: On the left hand side we have a simple parabolic function that only has one minimum. The red arrows show that picking a learning rate that is too big, can cause the network to never find the minimum. The same is shown on the right, where a more complex function contains two local minima, and one global minimum. The red arrow shows how a too big learning rate, or a too small learning rate (green arrows) can make the network converge to a local minima instead of the desired global minimum.

3.2.2.3 Optimizer

Related to the learning rate is the momentum. In this thesis two different **optimizers** have been tested; the Adam (Adaptive Moment Estimation) algorithm and the SGD (Stochastic Gradient Descent) algorithm. The former does not make use of a momentum, but the latter does.

- The **SGD** algorithm keeps the learning rate fixed throughout the whole training process, next to it being pretty inefficient for finding the global minimum, this also creates a high variance and oscillations. It can be thought of as the too big learning rate overshooting the minimum in Figure 3.12a. The ideal path would be to just follow the parabola until it reaches the minimum. In reality this will not happen and the path will have more of a zigzag structure, like the red arrows in the Figure. The momentum counters softens this zigzagging effect and keeps the learning process on track towards the global minimum by taking into account the previous step the learning process has taken in order to find the right direction (taking into account the exponentially weighted average). The SGD algorithm keeps the learning rate fixed throughout each cycle of training. In order to understand how the Adam optimization works, we need to first look at a different algorithm, namely the Root Mean Square Propagation (RMSprop) optimization.
- The **RMSprop** algorithm has the seem objective as the SGD method, namely dampening the oscillations and speeding up the convergence of the model. The difference lies in the method used to do achieve this goal. By computing an exponential moving average of the square of a gradient separately for each parameter we are training in the network, we can achieve the same goal as the momentum does, i.e., taking the past gradients into account when updating our model. The difference lies in the next step, where our step size with the newly calculated gradient is adjusted. This step size is now changed due to the exponential average we calculated in the previous step, by dividing the two by each other. This updates the initial learning rate and prevents the big zigzag pattern. Furthermore, when the model gets close to converging RMSprop will slow down the learning rate by performing simulated annealing.
- What the **Adam** optimizer does, is that it combines the SGD and RMSprop method into a new algorithm. If, for example, faced a problem with multiple local minima, as shown in Figure 3.12b, it will ensure there is minimum oscillation when using a gradient descent to find the global minimum, whilst at the same time ensuring the step-size is big enough to not get stuck in a local minimum. This gives the Adam algorithm a much higher performance, and also makes for a faster convergence of the model.

One might ask, why do we need to investigate the performance of the network with the SGD optimizer when, clearly, the Adam algorithm has so many advantages? Following the paper by (Wilson et al. 2017) they found that, when using adaptive gradient methods (like Adam), it is often the case that the network might give a very high prediction accuracy when training the network, however, when using a completely new test data set, it could perform very poor. This in contrast to the SGD method, which showed to have a much more robust generalization.

3.2.2.4 Loss Function and Scalers

The **loss function** is another strategy that can be used to make the network learn better. Since changing this strategy only applies to the feed forward neural networks (Section 3.2.3, it was decided to not include it in this thesis, since no comparison could be made. What we will investigate is the **scaler** used on the input and output data. Since the area we are training the network for is relatively small, some closer by stations will always dominate the input we give to the network, simply because the amplitudes will always be very high compared to the other, further away stations. This, and also the fact that not every earthquake will result in the same range of recorded amplitudes, might confuse the network. It is therefore important to scale the data beforehand. There are several built-in functions from the package Scikit-learn that could help us with these transformations. Examples of such built-in scalers are the MinMax scaler, and the Standard scaler. The former rescales the data to a certain range defined by us, and the latter removes the mean of the data and scales by the unit variance. Another possibility of scaling is by taking the maximum amplitude of each event that is fed to the network, and dividing the rest of the recording of this event by the maximum amplitude. This way information on the magnitude of the event is lost, but other information is not.

3.2.2.5 Epochs and Batch Size

Each network is trained for a certain amount of **epochs**. Here, an epoch means one cycle in which the network ’sees’ the complete training data set once. If we were to feed to complete data set at once to the network, compute the errors, and only then accordingly update the network, we would get a very generic update, which will probably does not do much for the convergence of our model. If we feed the samples one by one, calculating the error, and subsequently update the model per single sample, it would take a very long time for the network to converge. It is therefore more practical to feed the complete training set as batches to the network. If the **batch size** is small, the model will take longer to learn, but might be more accurate in the end, and if the batch size is too big, the network will learn nothing specific. Again, it is a fine line that we need to find in order to make all hyperparameters work together to get the best trained neural network possible.

3.2.2.6 Mixture of Gaussians

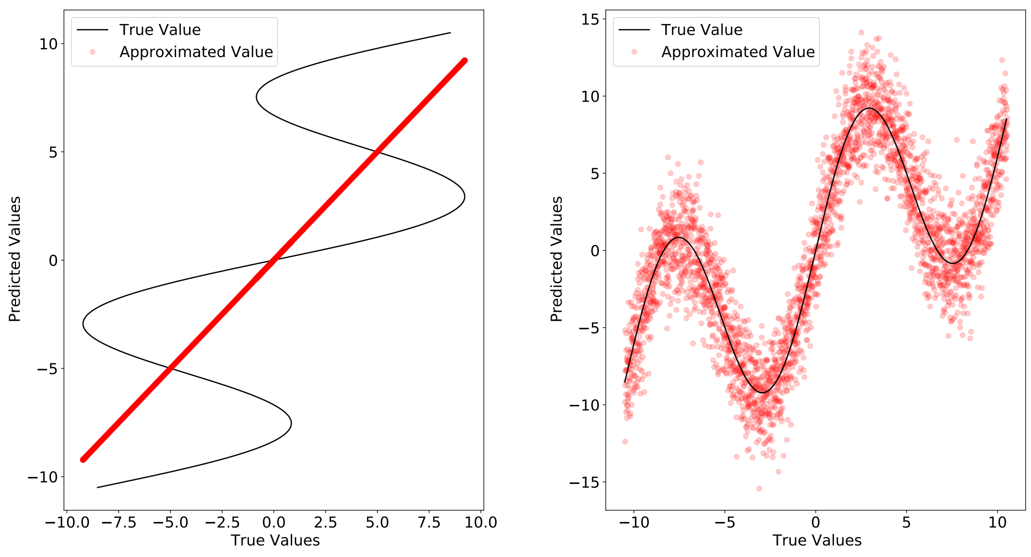
All hyperparameters and strategies, except for the loss function, described in Section 3.2.2 also apply to a mixture density network. We still have to check the activation function, number of hidden layers, neurons, the batch size, learning rate etc. The only thing that has changed with this switch of type of network is how we define the output of each prediction. As will be discussed in the next section (Section 3.2.3), our network does not predict the moment tensor values anymore, instead it describes the mean, variance, and weight of a specified number of Gaussians that together form a probability density function. The number of Gaussians set for the network to learn is therefore an added hyperparameter. Here, the rule is that too little mixtures will give a incomplete description of the data, whereas learning too many Gaussians to describe the probability density function, could lead to unnecessarily long run times. Despite being computationally more intensive, selecting a high number of Gaussians does not necessarily contribute to a poorer performance of the network. The Gaussians that contribute little to the overall probability density function will be set to almost zero via the weights that are assigned to each individual Gaussian by the algorithm.

Category	Parameter	What does it do?
	Hyperparameters	
Network Structure	Number of Hidden Layers	
	Number of neurons per layer	
	Activation function	Used to introduce non-linearity to the network.
Training Algorithm	Learning rate	How quickly the network updates the model
	Momentum	Only applies when using SGD
	Number of Epochs	How many times the complete data set is used to train the network
	Batch Size	The number of samples fed to the network each run within an epoch
	Strategies	
Regularization	Dropout	Technique to avoid over fitting by for excluding random neurons during training
	L1/L2 regularization	Penalizing neurons with a too great of a weight
Scaling	MinMax scaler	Rescaling the data within a certain range
	Standard scaler	Scaling to unit variance
	Custom Scaler	Scaling the complete input by the maximum amplitude
Optimization	ADAM	Updates model parameters using a stochastic gradient descent method that is based on estimations of the first- and second order moments
	SGD	Gradient descent optimizer

Table 3.3: A summary on the hyperparameters and strategies to be tried and tuned for a neural network. They are sorted per category, and, if needed, a short description of what the parameter defines is also given.

3.2.3 Mixture Density Network

A neural network aims to learn an underlying pattern that maps the input to the output target of our data set. Most of the time we see that a certain input x yields to a certain y within some range; we are dealing with an unimodal problem. But, what if, to simply put it, given a certain x, there are two y's possible that are not close to each other (hence, we are dealing with a multimodal problem)? Let us consider Figure 3.13. The left figure shows a wave-shaped function that can be mapped by a simple neural network. This forward problem should be easily learned by the neural network that was described in the previous section. However, if we take the inverse problem, the network will have a lot more trouble learning the s-shape (Figure 3.13b). What will happen is that the network will not know how to learn the underlying pattern anymore, since we are now dealing with a multimodal problem. It will therefore converge to the mean of the data set. This problem can be countered by using a mixture density network, as is shown in Figure 3.13b.



(a) Unimodal Problem (b) Multimodal Problem

Figure 3.13: In Figure 3.13a a sine function is approximated by a neural network by feeding a certain x to the network that then predicts the y value. The network can anticipate the underlying pattern pretty well. If we approach the inverse problem of Figure 3.13a, we would obtain Figure 3.13b. In Figure 3.13a a certain value of x would have one value of y possible, within a certain range. Now, however, we see that a certain value of x can yield multiple values for y. This confuses the neural network, and instead of trying to predict the underlying pattern, it will converge to the mean of the data set.

As said before, if a conventional neural network tries to learn the function visualized in Figure 3.13b it will converge to the mean of the data set. Or, to be more specific, it will learn the conditional average as a function of our input, and the average variance (thus, the range around the true values as observed in 3.13a) of the data. Obtaining these averages of a certain function will work for most cases, however, here we would need a bit more information on the distribution of the function in order to make some educated guess. As (Bishop 1994) puts it in his paper on mixture density networks; a conventional neural network solving a certain classification problem works very well, but as soon as we are trying to predict continuous problems (solve a regression problem), the conditional average that we learn will be very limited. Next to that, we do not know anything about the uncertainty with which a prediction is made. It is therefore in this paper that Bishop proposes the first ideas on mixture density networks and the theory behind it.
To give a simpler version of how the theory works, let us consider first again the conventional neural network. We could estimate the conditional average of the data set and its average variance. These two parameters allow us to construct a probability density function for each of the output values of the network with the given input. Note that the average variance here is a non changing value for the complete distribution of the data set and that with the conditional average only one Gaussian-shaped probability density function will be estimated. So what if we were to replace this Gaussian distribution with a mixture model? With this we mean that instead of using only one Gaussian, we let a finite number of Gaussian make up the probability density function. This can be written in the form

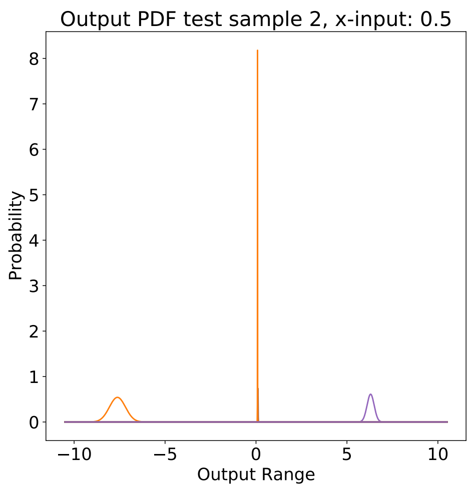
$$p(\boldsymbol{t}|\boldsymbol{x}) = \sum_{i=1}^{m} \alpha_i(\boldsymbol{x})\phi_i(\boldsymbol{t}|\boldsymbol{x}) \tag{3.6}$$

Here, α represents the mixing coefficient of each individual component in the mixture (with a total of m), and ϕ representing the conditional density function with output target **t** for an input **x**. As said before, the distribution of the probability is taken to be Gaussian. We can therefore represent the conditional density ϕ as a kernel function of the Gaussian form

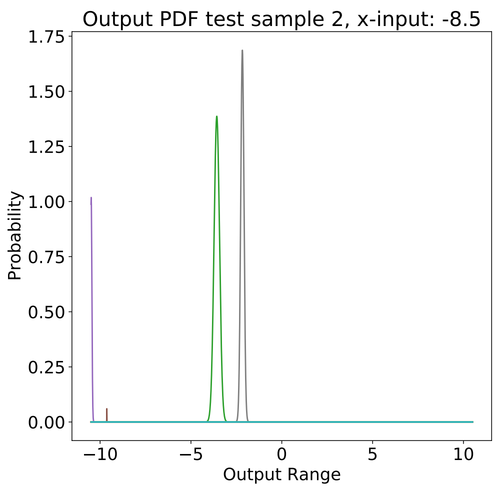
$$\phi_i(\boldsymbol{t}|\boldsymbol{x}) = \frac{1}{(2\pi)^{c/2}\sigma_i(\boldsymbol{x})^c} exp\{-\frac{\|\boldsymbol{t}-\boldsymbol{\mu}_i(\boldsymbol{x})\|^2}{2\sigma_i(\boldsymbol{x})^2}\} \tag{3.7}$$

with μ representing the mean of each Gaussian, and σ the variance. Both Equation 3.6 and 3.7 do not need the target output **t** to be independent of each other. This is in contrast with the single Gaussian representation that we see with a conventional neural network.

(Binder 2018) states that the problem at hand should be solvable by using a simple feed forward neural network, i.e., there is no indication that our problem is mutlimodal. So, why are we then looking into the mixture density networks? It is because they can give us more insight into the uncertainty of each prediction made by the network. By not only evaluating the variance score of the network, but also looking at the probability distribution for each event we put into the network, we could make a pretty accurate estimate of the uncertainty. To illustrate this, let us go back to the toy example that we looked at in the beginning of this section. As we can see, the network has made a pretty good prediction on the actual shape of the function we are trying to map, but what is the uncertainty when making these decisions? Let us consider Figure 3.14, where the probability density function of two test samples fed to the trained network from Figure 3.13a is given. In Figure 3.14a the network is very sure of its own prediction, however, Figure 3.14b shows that given a different value for the input, the network has some more issues being accurate.



(a) Good prediction



(b) Less sure prediction

Figure 3.14: In Figure 3.13a a sine function is approximated by a mixture density network. If we were to make a new test set, i.e., generate some new input x that is fed to the network, which in turn predicts the optimal output y, the mixture density network will not only give us the best predicted value, but also the uncertainity of its choice. This is for example shown in Figure 3.14a, where the network is fairly sure that given an input of 0.5, the output should be 0. However, if we feed the network an input of 8.5, it still shows us what the best prediction for the output would be (the purple Gaussian), however, looking at the other two Gaussians drawn left of the middle Gaussian (green and pink Gaussian), it is seen that there is some chance that the output could actually be different.

The Feed Forward Neural Netowrk (FFNN) gives us the six Moment Tensor (MT) components as an output. This is contrary to the Mixture Density Network (MDN) network, where we will obtain the means, weights, and variances of the probability distribution function (PDF) from which we can draw the best possible prediction. To train the network, however, our input for the training labels, i.e., the true values of the moment tensor, is the same as for the input of the FFNN. How do we account for this change in output, but not of input values? If we make some minor changes to the loss function, this problem could be easily solved. By constructing a mixture model from the output of the network, the logarithmic probability mass function can be constructed that relates to the true values of the input. The outcome of this is then used as a means of updating the network.

3.3 Feed Forward Neural Network vs. Mixture Density Network

In Section 3.2.1 we talked about the general theory that forms the basis for the FFNN and the MDN. Section 3.2.3 explained the difference of a MDN to a FFNN, but did not really elaborate on how these two separate types of network can complement, or compete against each other. We could ask: Why not only use a MDN always? It gives an estimation of the moment tensor and also the associated uncertainties that come with each prediction. A short review of the literature can give us a first answer to this question.

- (Binder 2018) solves the problem, using a simple model, testing the performance only with a FFNN, which is fast to train, and does not cost to much computational power, but, it does not give us any information on the uncertainity of each prediction.
- (Steinberg et al. 2021) proposes the use of a Bayesian Neural Network to assess the uncertainties in the networks prediction. Using 128 CPUs the creation of the synthetic data, and the training of the network took them three months.
- (Kuang et al. 2021) makes use of a more complex network structure that uses as training labels three Gaussian probability distributions that represent the strike, slip and dip of the corresponding input, thus taking it a step further than (Steinberg et al. 2021), where the training labels are the focal mechanisms itself. This FMNet takes around 5 hours to train, using four GPUs of NVIDIA Tesla V100.
- In his dissertation, (Käufl 2015) not only estimates the focal mechanism of each event, but also the depth, location, logarithmic magnitude, and the half-duration correlated to the size of the event, using a mixture density network that follows the network architecture of a simple feed forward neural network. Depending on the type of network training(i.e., the length of the time window used), the CPU hours needed to construct a good predicting network varies between a 100 and 10000 hours.

On the one hand we have the paper by (Binder 2018) that tells us a network should be trainable with not too many resources, whereas on the other hand we have the papers by (Steinberg et al. 2021), (Kuang et al. 2021), and (Käufl 2015), that all advocate for the MDN, however, the network will take a longer time to train and is computationally more expensive, see Figure 3.15. Feeling that the gap in computational resources needed and the difficulty of the network is too big between the latter and former three papers, it was decided to try out how the prediction precision of both the FFNN and MDN would differ, if using the exact same model to train the algorithms. Do they need different network architectures to perform better, is there a big contrast in computational time needed to train the network, and does one network perform better than the other one?
Training the network is done on the Lisa cluster computer. This server has four types of CPU's, of which we only used two, assigned by availability. For specifications see Table 3.4. This brings us to the way we will test the differences between the FFNN and the MDN. For each network type, three types of models will be tested; a homogeneous model without noise with fixed source location, a heterogeneous model without noise with varying source location, and the same heterogeneous model with varying source location with added Gaussian noise. An overview of these three models can be found in Figure 3.16.

	CPU	
	Intel® Xeon® Gold 6130 Processor	**Intel® Xeon® Silver 4110 Processor**
Processor Base Frequency	2.10 GHz	2.10 GHz
Memory	96 GB UPI 10.4 GT/s	96 GB UPI 9.6 GT/s
Cores	16	16

Table 3.4: Two types of CPUs used for this thesis

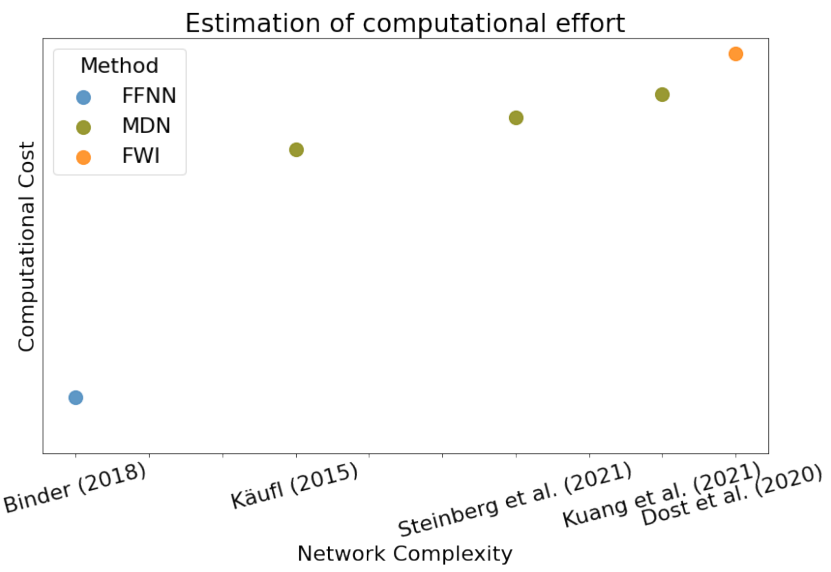


Figure 3.15: A rough estimation of the computational cost versus the network complexity of the four different papers discussed in the text. It is seen that there is a gap of research that investigated both the MDN and FFNN network with simpler network architectures.

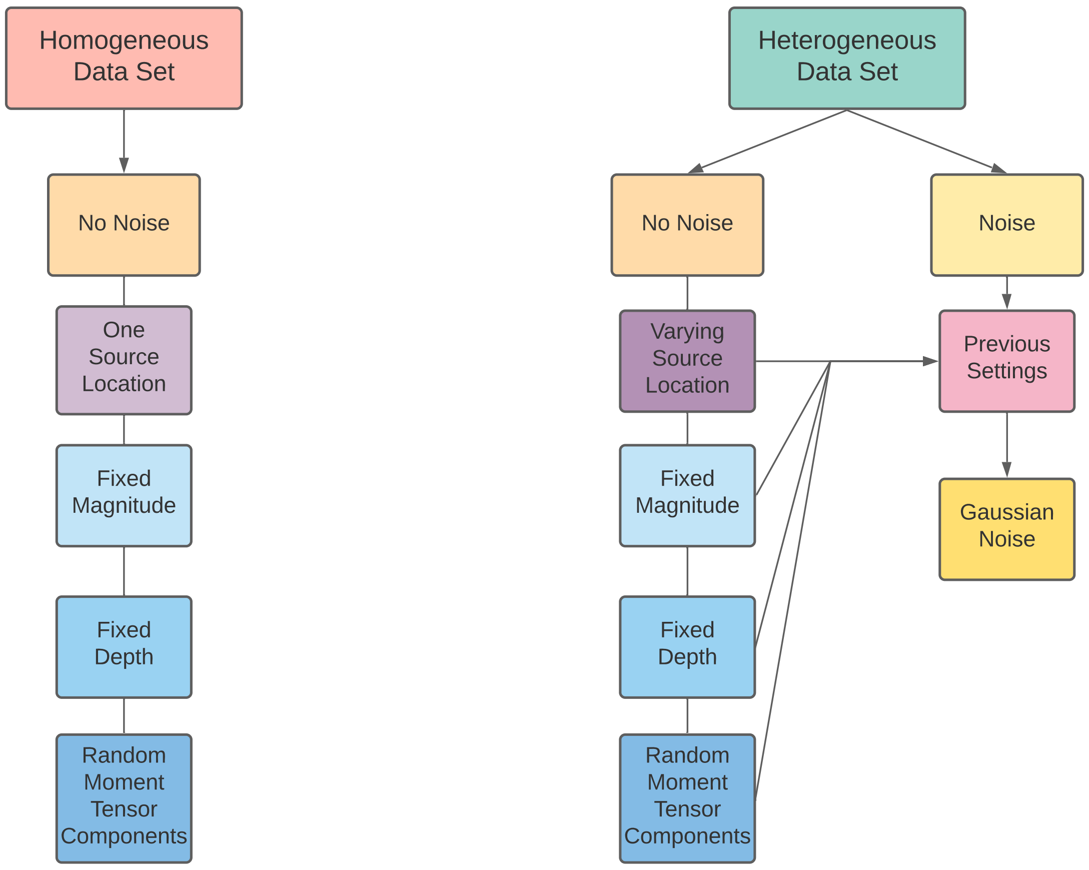


Figure 3.16: Overview of the models to be tested by the neural network.

Chapter 4

Results

In this Chapter we will walk through the process of finding the right type of neural network, the changes made to the data set in order make the problem more complex, and to learn more about the hyper parameters and how to fine tune them. To give a short refresher on the hyperparameters and strategies tested in this thesis, we refer the reader to Table 3.3 in Section 3.2.2.
As we progress through this chapter, it will become evident that there is not a lot of difference between the accuracy of the individual moment tensor component predictions for the simpler data sets, however, when using the more complex data sets (the heterogeneous velocity model) we will see that some settings will work better for prediction of some components than others. The moment tensor contains nine components, of which six are independent. From now on, if the term 'first moment tensor component' or 'MT1' is used, we mean the mnn or $M_x x$ component of the moment tensor. An overview of this renaming can be found in Figure 4.1. The diagonal components of the moment tensor represent the linear vector dipoles, i.e., it represents the tensile or compressive faulting. As (Steinberg et al. 2021) mentions, when testing a model that contains errors, it is exactly the CLVD and isotropic components that give rise to the highest uncertainties for their network. (Kuang et al. 2021) reports errors in the rake angle of up to twenty degrees, together with a low prediction probability. Both of these papers use strike, dip, and rake angles as labels. This thesis will deal only with the six independent moment tensor components, since it was more convenient in relation with the creation of the synthetic data set, but also because it gives the possibility to investigate the full moment tensor earthquakes. However, as (Dahm & Krüger 2014) mentions, the isotropic component, given certain data, might be difficult to resolve.

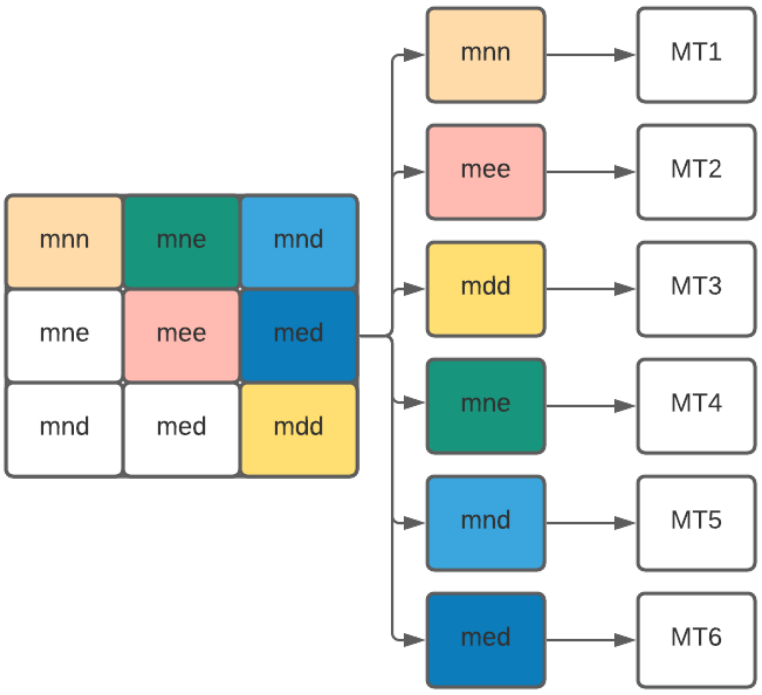


Figure 4.1: Renaming of the MT components. Here, MT1 to MT3 represent volumetric changes, or the isotropic component of a source. If only this diagonal contained non-zero values, the corresponding source would be a pure explosion (positive values) or implosion (negative values). MT4 to MT6 represent the force couples with moment. These off diagonal components are associated with rotational movement of the source.

4.1 Feed Forward Neural Network

The accuracy and efficiency of using a multi-layer perceptron to estimate the six moment tensor components was investigated using different data sets. Two different models are used to understand the impact of feeding the neural network with data sets of different sizes and heterogeneity. The initial process of fine tuning the hyperparameters was identical for each data set tried. After gathering info on which settings or strategies work best for each type of data, the amount of data needed to obtain satisfying results will be investigated as well. As we will see, is that each change to the data set, may it be the velocity model, or the amount of data used to train the network, will yield a different combination of hyperparameters that makes the algorithm perform best. The results in this chapter describe the performance of the neural network on a data set that it has never seen before. This is the so-called test set, and will be kept apart from the train set until the learning of the algorithm has converged. For example, in this thesis a data set of 10.000 earthquakes is used to investigate the prediction of the moment tensor. This means that eighty percent of this data is used to train and validate the network, and the remaining twenty percent is used to test the trained network.
The feed forward net only outputs the estimated moment tensor values. It therefore does not tell us anything on the uncertainty of its prediction. With so many parameters to be fine tuned and predicted, it is sometimes easy to lose the overview. It is for this reason that only a selection of the results will be shown here, the rest can be found in the Appendix B. Mainly the results for one of the moment tensor components will be shown, unless stated otherwise.

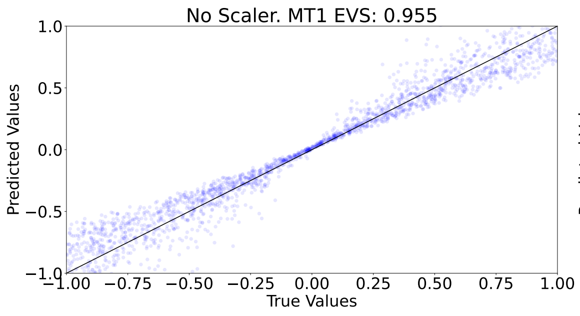
4.1.1 Homogeneous Model

The first data set considers using a homogeneous velocity model as shown in Figure 3.2a. This particular data set has a fixed magnitude, depth, and source location, in order to evaluate the first performances of the neural network.

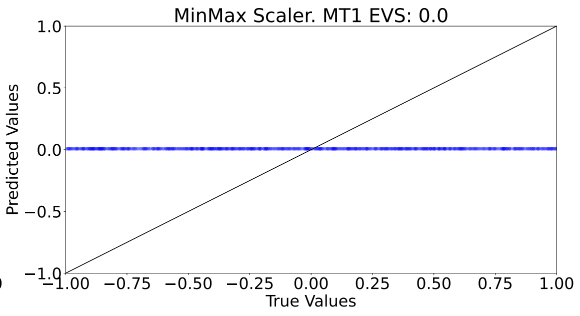
First, we will evaluate how scaling the input and target values of the data set could benefit the learning of the network. Four options of scaling the input data are shown in Figure 4.2, namely using no scaler, a MinMax scaler, a standard scaler, and scaling based on the maximum amplitude. What each of these scalers exaclty do is explained in Chapter 3, Section 3.2.2. The main idea behind standardizing, i.e. scaling, the data is to ignore outliers. The MinMax scaler and standard scaler both remove the mean of each sample that is to be scaled, and dividing their features by the standard deviation of the data set. The data set used in this thesis is uniformely distributed, which could in theory cause problems with the scalers, since they perform better with a Gaussian distribution.
Sometimes, it also lends itself when a scaler is applied to the target values used during the training to correct the network. An example of how the scaling of the target values influences the performance, we refer the reader to Figure 4.3. It is seen that for the homogeneous data set, the network does not perform better or worse with a changing scaler, except for a scaling of zero to one. Figure 4.3b might seem like something went wrong when rescaling the predicted moment tensor components back to the initial range of negative one to positive one, however, if we look at the explained variance score of this data set, we can see that that is very low as well. Since the same scaler does seem to work for more complex models, it is not exactly clear as to why it performs so poorly on the simpler model.

Hyperparameter / Strategy	Value / Type
Number of hidden layers	1
Number of neurons per layer	100
Activation Function	elu
Learning Rate	0.001
Batch Size	750
Regularization	Dropout (0.05)
Initialization	-
Scaler	Input scaler varies
	Target values scaler varies
Optimization	ADAM
Size data set	Varying

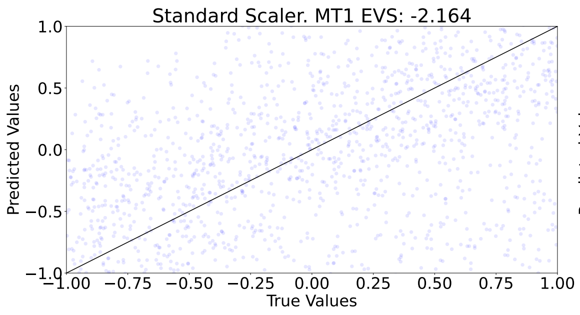
Table 4.1: Network architecture of the net used to evaluate the performance of the algorithm with changing input and targer scalers. Results can be found in Figure 4.2 or 4.3.



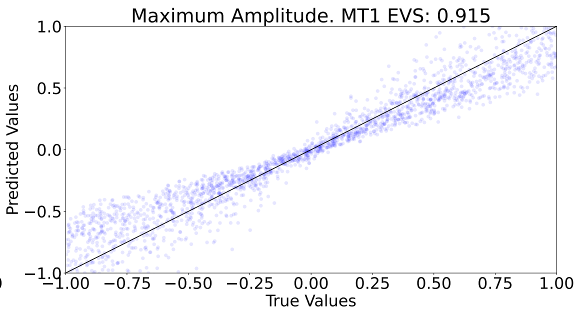
(a)



(b)

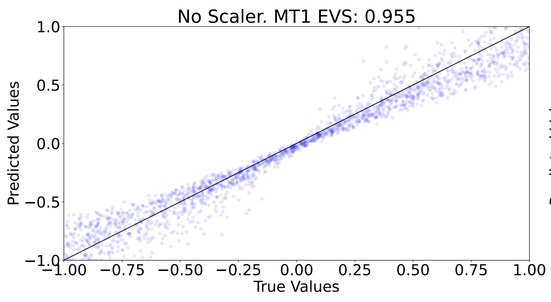


(c)

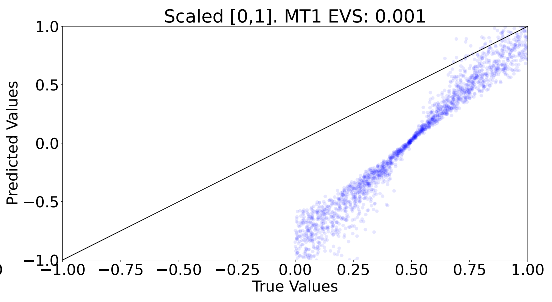


(d)

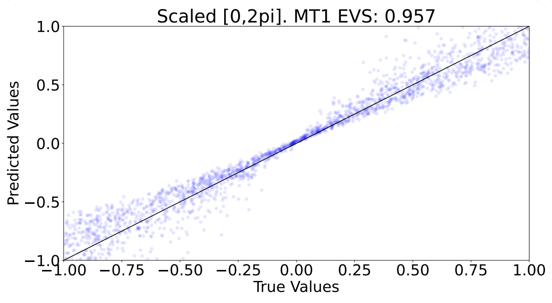
Figure 4.2: The performance of a network when only changing input data scaler used to train the algorithm measured by visualizing the true versus predicted values of the test set and the explained variance score (EVS). The four plots show the explained variance score for the first moment tensor, where each blue dot depicts a test sample. We see that the MinMax- and Standard Scaler make the network predict very poor. It is seen that using no scaler on the input data might work the best in case of the homogeneous model. The black linear line denotes the trend a perfect predictor should have; each predicted value being exactly the same as the true value. The network architecture for this particular problem can be found in Table 4.1.



(a)



(b)



(c)

Figure 4.3: The performance of a network when only changing the target scaler used to train the algorithm measured by visualizing the true versus predicted values of the test set and the explained variance score (EVS). The four plots show the explained variance score for the first moment tensor, where each blue dot depicts a test sample. In the case of a homogeneous model we see that there is not really a scaler needed for the network to predict the moment tensor components accurately. The black linear line denotes the trend a perfect predictor should have; each predicted value being exactly the same as the true value. The network architecture for this particular problem can be found in Table 4.1.

Another point of interest to show is how the amount of hidden layers, and the number of hidden neurons used can influence the performance of our network. Our input layer is quite large (over 9000 neurons big), and compressing it to a much smaller size of neurons for the first hidden layer might feel like stacking the data, as is done in signal processing. In some ways this is true; we force the network to only extract the most important features and translate this to the moment tensor. We can see in Figure 4.4a that using too little neurons in the hidden layer gives an unsatisfactory result, especially if we compare the outcome to Figure 4.4b. This means that we the number of important features we allow the network to extract are too little for the algorithm to make sense out of it. Then, if we add an extra hidden layer to the network architecture, we can see that the network can explain the variance even better (Figure 4.4c). By setting the first hidden layer to have less neurons than the second, the network will only extract the most important features of the input data, before it 'rebuilds' these features to a bigger set of neurons in the second layer. This second layer can then elaborate on what these specific features mean exactly to the prediction of the moment tensor, thus yielding a higher EVS.
In the case of data set based on the homogeneous velocity model using too little neurons in the hidden layer luckily does not show that much difference yet, but still it is noteworthy to keep in mind how the amount of neurons influences the performance of our network.

Hyperparameter / Strategy	**Value / Type**
Number of hidden layers	Varying
Number of neurons per layer	Varying
Activation Function	elu
Learning Rate	0.001
Batch Size	750
Regularization	Dropout (0.05)
Initialization	-
Scaler	Input divided by maximum amplitude
	Target values scaled from [0,2pi]
Optimization	ADAM
Size data set	10.000

Table 4.2: Network architecture of the net used to evaluate the performance of the algorithm with changing size of train data. Results can be found in Figure 4.4.

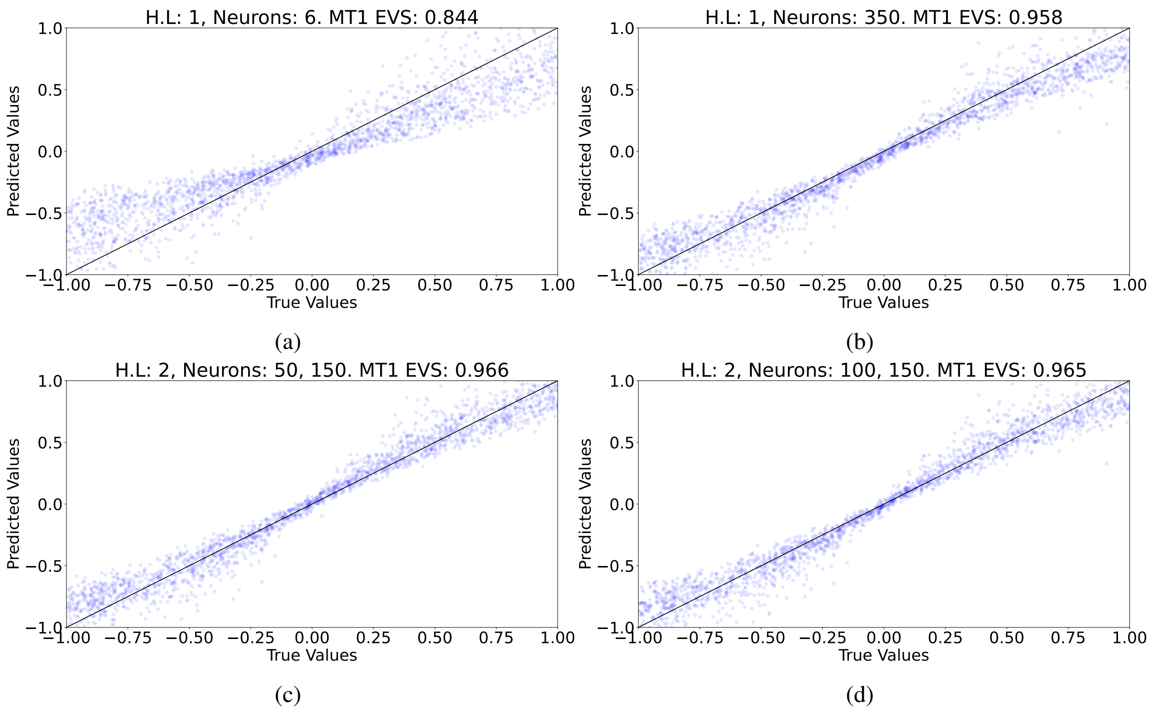
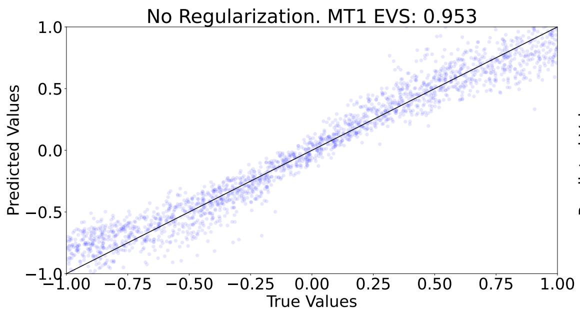


Figure 4.4: The performance of a network when only changing the amount of hidden layers and/or neurons used to train the algorithm measured by visualizing the true versus predicted values of the test set and the explained variance score (EVS). The four plots show the explained variance score for the first moment tensor, where each blue dot depicts a test sample. As can be seen is that the explained variance score almost reaches the maximum of one in some of the figures (Figure 4.4b, 4.4c, and 4.4d). This is vary favorable, since it means that there is little difference between the predicted and true values. The black linear line denotes the trend a perfect predictor should have; each predicted value being exactly the same as the true value. As can be seen is that the blue dots follow this line pretty closely. The network architecture for this particular problem can be found in Table 4.2.

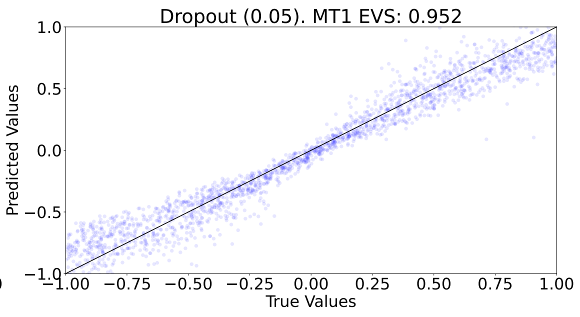
Lastly, let us take a look at the influence of the different regularizers we can use to constrain overfitting on the train data. Figure 4.5 shows that there is almost no difference between the regularizer used for this problem. This should not be that surprising, since we are still dealing with a homogeneous data set that does not contain any noise at all. Regularization is mostly only needed when noise is present in a data set, when the data is complex, or when there the training set is not sufficient enough to represent the whole problem. In the last case the regularization would counter the overfitting of the network on the unrepresentative data set.

Hyperparameter / Strategy	**Value / Type**
Number of hidden layers	1
Number of neurons per layer	100
Activation Function	elu
Learning Rate	0.001
Batch Size	750
Regularization	Varying
Initialization	-
Scaler	Input divided by maximum amplitude
	Target values scaled from [0,2pi]
Optimization	ADAM
Size data set	10.000

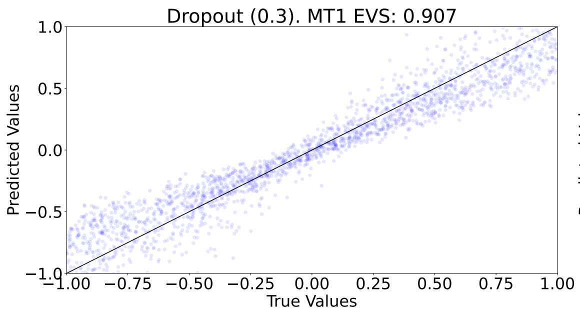
Table 4.3: Network architecture of the net used to evaluate the performance of the algorithm with changing regularizers. Results can be found in Figure 4.5.



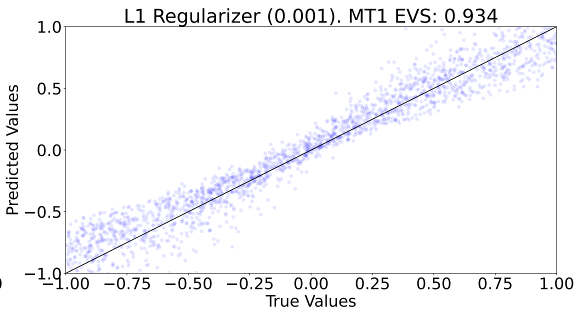
(a)



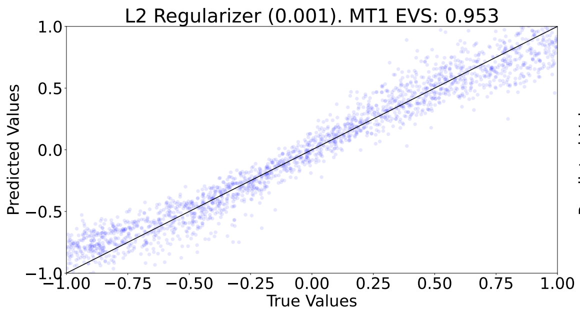
(b)



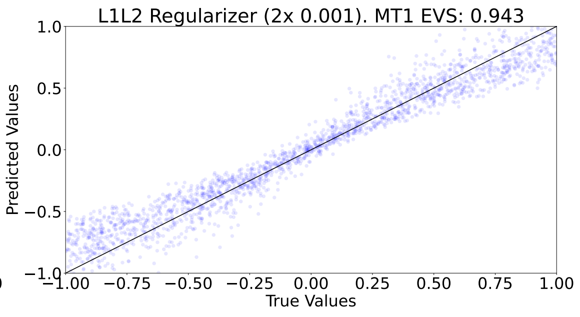
(c)



(d)



(e)

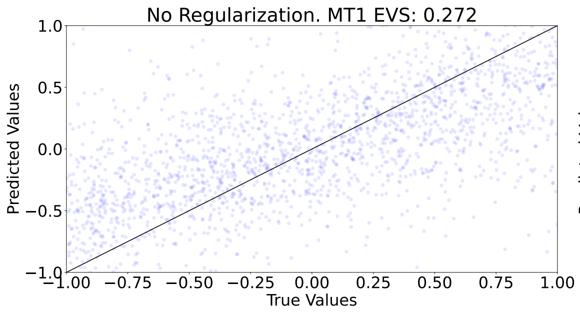


(f)

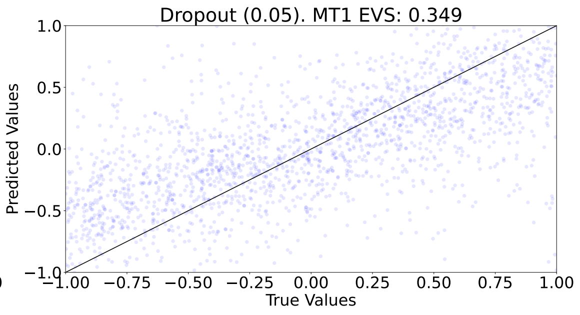
Figure 4.5: The performance of a network when only changing the type of regularizer used to train the algorithm measured by visualizing the true versus predicted values of the test set and the explained variance score (EVS). The six plots show the explained variance score for the first moment tensor, where each blue dot depicts a test sample. The black linear line denotes the trend a perfect predictor should have; each predicted value being exactly the same as the true value. As can be seen is that the blue dots follow this line pretty closely, even when no regularization is used. The network architecture for this particular problem can be found in Table 4.3.

4.1.2 Heterogeneous Model

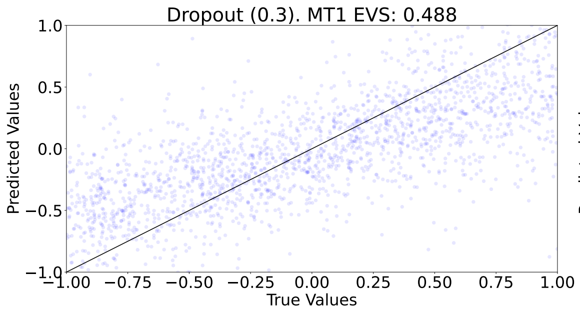
An early conclusion that can be made from the previous section is that the feed forward network can predict the moment tensor components pretty well with explained variance scores going up to 0.96 given the simple model. Of course we do not know yet the uncertainty that is related to these predictions; we will find this out in Section 4.2.1. Let us first look at how the accuracy of our predictions change when we switch to a more complex model, namely the locally averaged 1D model of the Groningen subsurface (as shown in Figure 3.2b), but leave the noise out. It is already with this data set that we will see that the network does not predict each moment tensor with the same accuracy anymore and the we need to make use of a scaler to get reliable results. If we take the same network architecture from Table 4.3 to test how the different regularizers influence the accuracy of the network and would only look at the explained variance score of the first moment tensor component, we could say that the network is not predicting that well. However, looking at, for example, the prediction of all moment tensor components using the best performing regularizer of Figure 4.6 (Figure 4.6f), a different light can be shed on the accuracy of the predictions. It is in Figure 4.7 that we can see how the network has a different EVS when it comes to the individual moment tensor components.



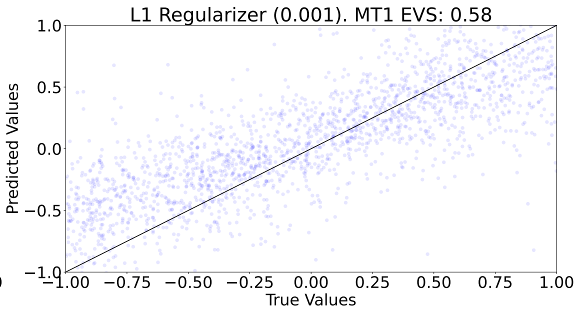
(a)



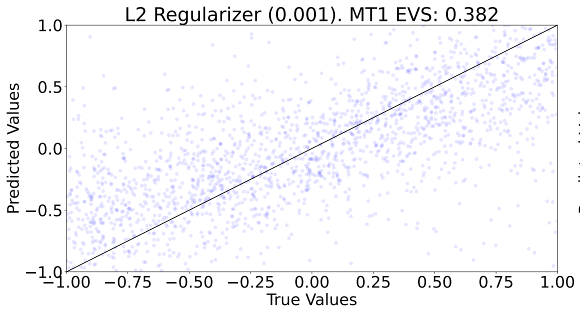
(b)



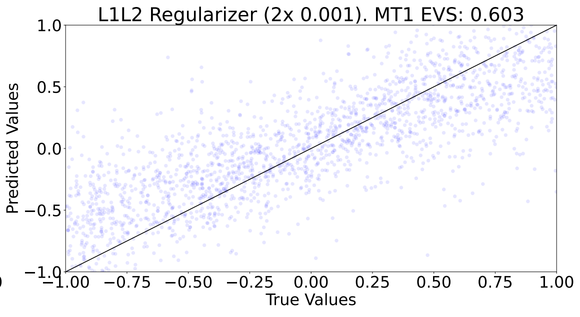
(c)



(d)



(e)



(f)

Figure 4.6: The performance of a network when only changing the type of regularizer used to train the algorithm measured by visualizing the true versus predicted values of the test set and the explained variance score (EVS). The six plots show the explained variance score for the first moment tensor, where each blue dot depicts a test sample. The black linear line denotes the trend a perfect predictor should have; each predicted value being exactly the same as the true value. Contrary to the case were the homogeneous data set is used, regularization is definitely needed in order to train the network. Figure 4.6f shows the highest explained variance score of all types of regularization tested in this figure. This score is still not a very good score. The network architecture for this particular problem can be found in Table 4.3.

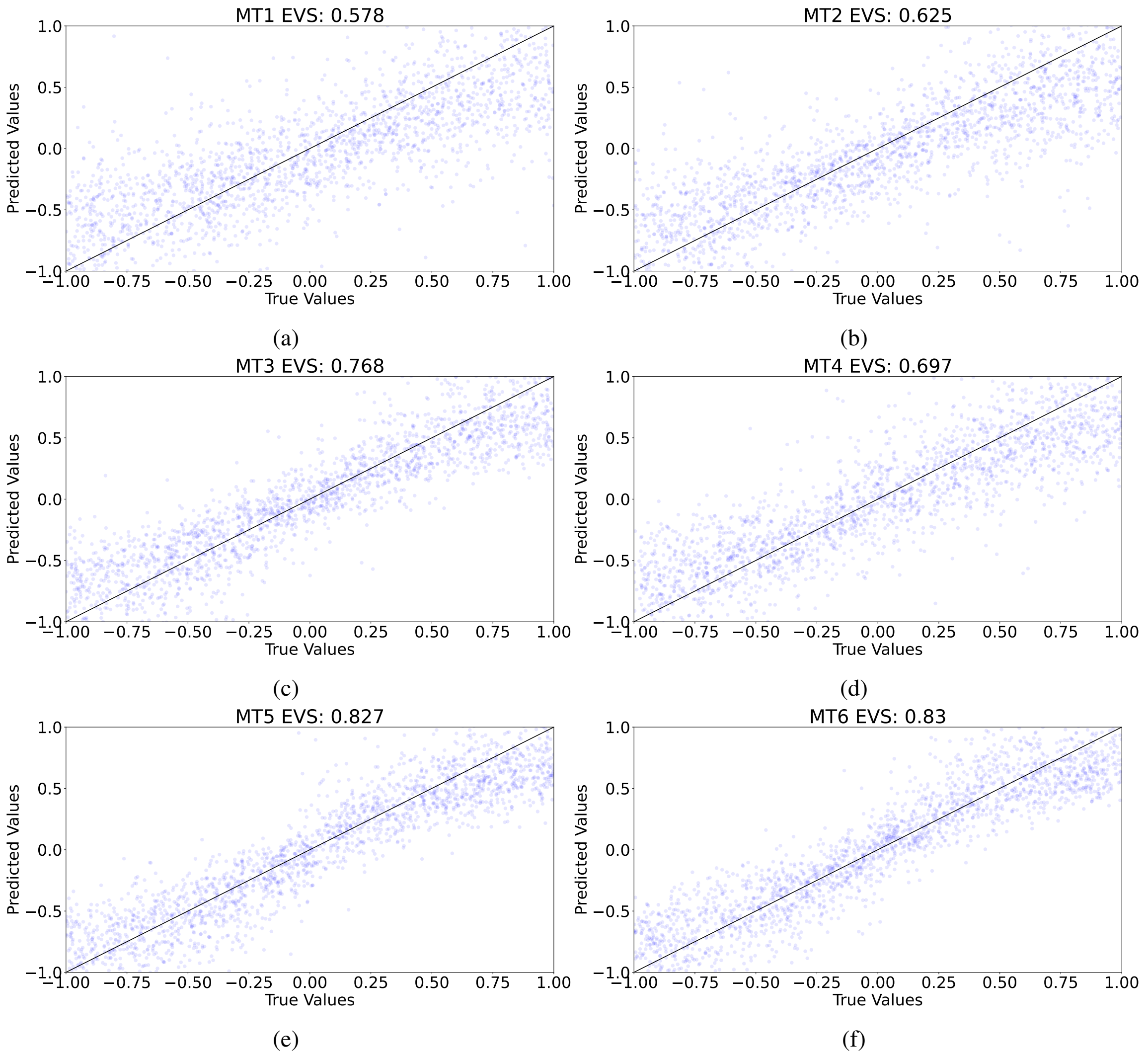
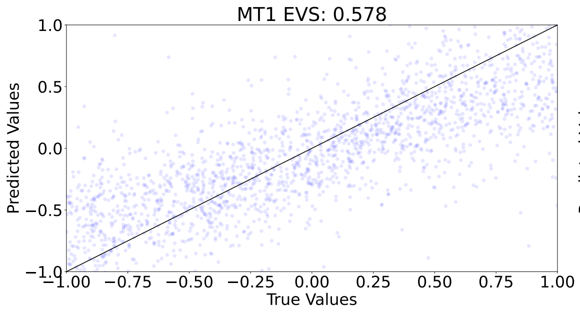


Figure 4.7: The performance of a network when using a L1L2 (2x 0.001) regularizer to train the algorithm measured by visualizing the true versus predicted values of the test set and the explained variance score (EVS). The six plots show the explained variance score for each individual moment tensors component, where each blue dot depicts a test sample. The black linear line denotes the trend a perfect predictor should have; each predicted value being exactly the same as the true value. As can be seen is that the blue dots follow this line more closely for some components than others. Especially moment tensor components five and six show a good predictability. The network architecture for this particular problem can be found in Table 4.3.

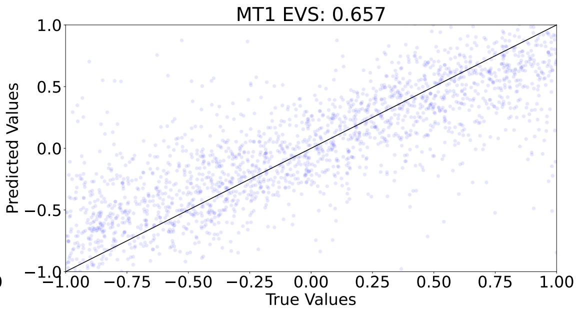
If we look at the above results, we can see that the predictions for the first three moment tensor components are a lot poorer than the other predictions. A possible explanation could be that another combination of hyperparameters and strategies in the network's architecture might lead to a more even accuracy of prediction for the moment tensor components. We can test this via Figure 4.8a, which shows the EVS and true versus predicted values for the first moment tensor component of the best performing network of Figure 4.6. Figure 4.8b shows the performance on the first moment tensor component of a network with the architecture shown in Table 4.4. The network with a dropout performed not so well in Figure 4.6, however, with slight changes to the network's architecture we can get a EVS that is a bit better than that of the network with the $L1\_L2$ regularizer.

Hyperparameter / Strategy	Value / Type
Number of hidden layers	1
Number of neurons per layer	50
Activation Function	elu
Learning Rate	0.001
Batch Size	750
Regularization	Dropout (0.05)
Initialization	-
Scaler	Input divided by maximum amplitude
	Target values scaled from [0,2pi]
Optimization	ADAM
Size data set	10.000

Table 4.4: Network architecture of the net used to evaluate the performance of the algorithm with changing regularizers. Results can be found in Figure 4.8b.



(a) $L1\_L2$ (2x 0.001) regularizer



(b) Dropout (0.05) regularizer

Figure 4.8: The performance of a network when using a L1L2 (2x 0.001) regularizer to train the algorithm measured by visualizing the true versus predicted values of the test set and the explained variance score (EVS), set against the same performance when using a dropout rate and a different network architecture. The two plots show the explained variance score for the first moment tensor component, where each blue dot depicts a test sample. The black linear line denotes the trend a perfect predictor should have; each predicted value being exactly the same as the true value. The network architecture for Figure 4.8a can be found in Table 4.3, for Figure 4.8b refer to 4.4.

In spite of having shown that different architectures for our algorithm can lead to a slightly better performance, the EVS is still pretty poor. Interestingly, this poor EVS is again only valid for the first, second, and third components, whereas the fourth, fifth, and sixth moment tensor components give accuracies equal to those in Figure 4.7. There might be another explanation of this behaviour, namely that we need to design six separate networks that each are tailored to their own individual moment tensor component. Although using a different type of network (a MDN) (Käufl 2015) also reports that independently estimating the desired outputs, can lead to differences in network architecture and better performances overall. Figure 4.9 tries to show this. As can be seen is that even with different network architectures, the network still performs poor on especially the first, second, and fourth moment tensor components. If we look back at the definition of the moment tensor components and what forces they describe (Figure 2.6 and Figure 4.1), observe that moment tensor components one to three all represent the diagonal tensor components. These represent volumetric changes, whereas the off-diagonal components speak for the rotational effects (moment tensor components three through six). It might be that the network has a harder time detecting these volumetric forces, since they might change the trace less.

Hyperparameter / Strategy	**MT1**	**MT2**	**MT3**	**MT4**	**MT5**	**MT6**
Number of hidden layers	1					
Number of neurons per layer	100	100	100	100	300	200
Activation Function	elu					
Learning Rate	0.001					
Batch Size	1500	500	1500	250	1500	1500
Regularization	Dropout (0.15)					
Initialization	-					
Scaler	Input divided by maximum amplitude					
	Target values scaled from [0,2pi]					
Optimization	ADAM					
Size data set	10.000					

Table 4.5: Network architecture of the net used to evaluate the performance of the algorithm with changing regularizers. Results can be found in Figure 4.9.

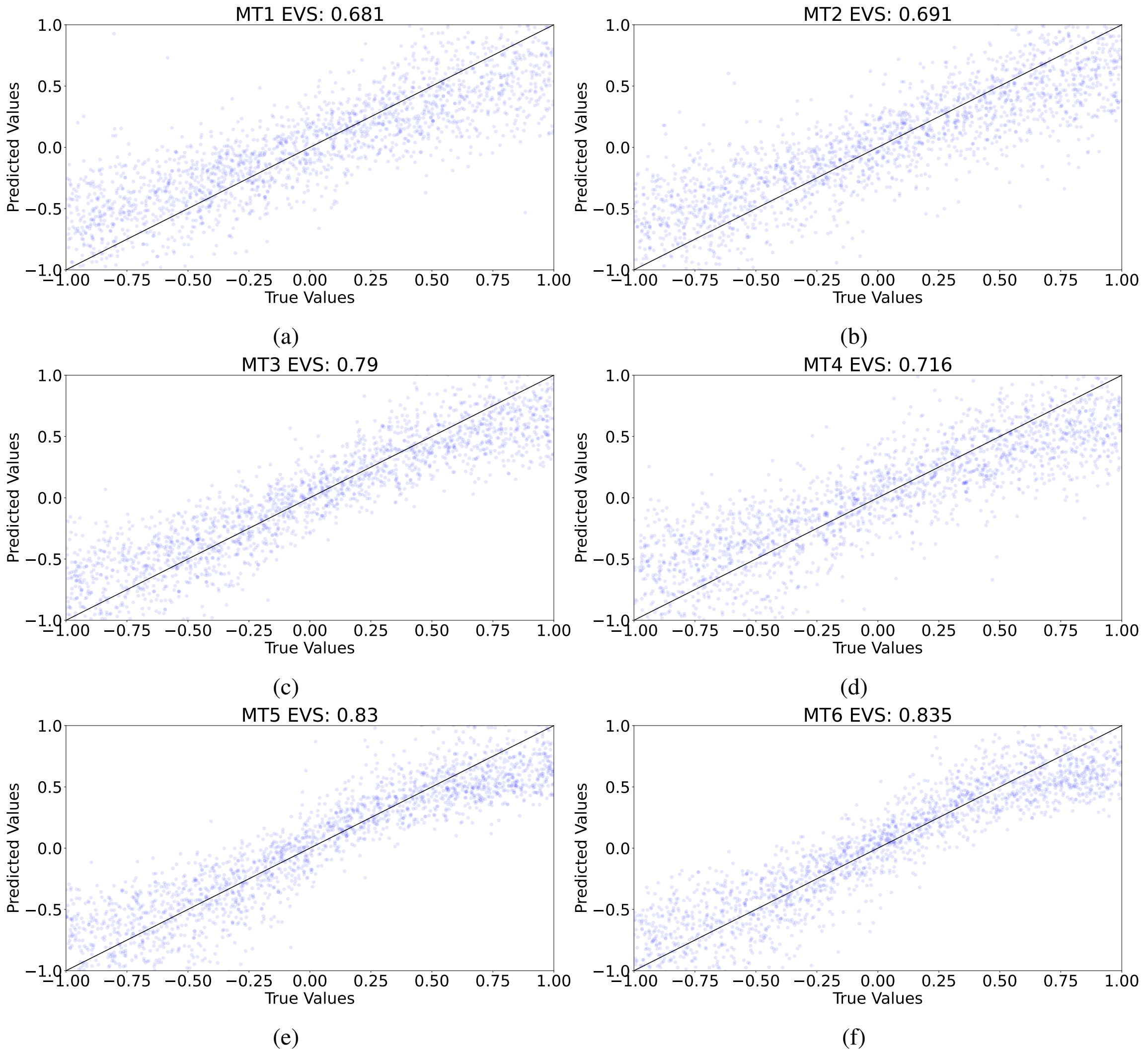
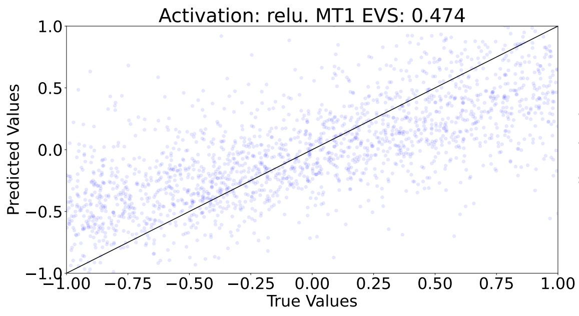


Figure 4.9: Results of the prediction of the moment tensor components, where each moment tensor components is estimated by an unique network visualized by the true versus predicted values of the test set and the explained variance score (EVS). The six plots show the explained variance score for each individual moment tensors component, where each blue dot depicts a test sample. The black linear line denotes the trend a perfect predictor should have; each predicted value being exactly the same as the true value. As can be seen is that the blue dots follow this line more closely for some components than others. Especially moment tensor components five and six show a good predictability. The network architecture for this particular problem can be found in Table 4.5.

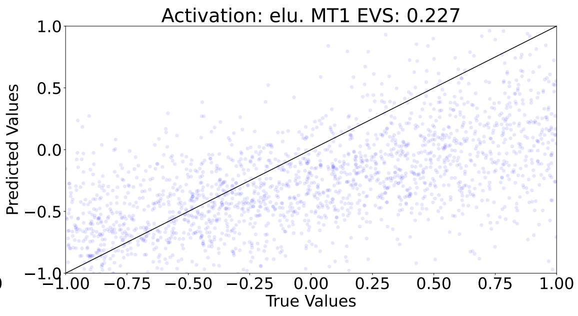
Lastly, we present the effect of a changing activation function has on the performance of the network, see Figure 4.10. Here, a mix of the four different activation functions explained in Chapter 3 will be shown; the elu, selu, and relu activation functions, mostly used for regression problems, but also the effect of using a sigmoid activation function, which is more tailored to classification problems. This is mostly valid for the activation function in the output layer, and for regression problems we always use a linear function as the output activation function. Still, it is uncommon to use a sigmoidal activation function in the hidden layers of a network, since it has this problem of vanishing gradients. The vanishing gradient can be explained by looking at the derivative of the sigmoid, which translates a big change in data to a very small gradient, making it possible to vanish. An explanation of the good performance of the sigmoid function might be that the changes in the traces are not that big anyways, thus causing the sigmoidal activation function working in the favor of the training on the network. The activation function used so far (the elu function) performs a lot poorer compared of the other activation functions shown in Figure 4.10. Again, this is another proof of having multiple optimal combinations of strategies and hyperparameters that can cause the network to perform a lot better. But, in general, it was seen that networks that make use of the sigmoidal or selu activation function perform slightly better than others.

Hyperparameter / Strategy	**Value / Type**
Number of hidden layers	1
Number of neurons per layer	50
Activation Function	varying
Learning Rate	0.001
Batch Size	750
Regularization	Dropout (0.2)
Initialization	-
Scaler	Input divided by maximum amplitude
	Target values scaled from [0,2pi]
Optimization	ADAM
Size data set	10.000

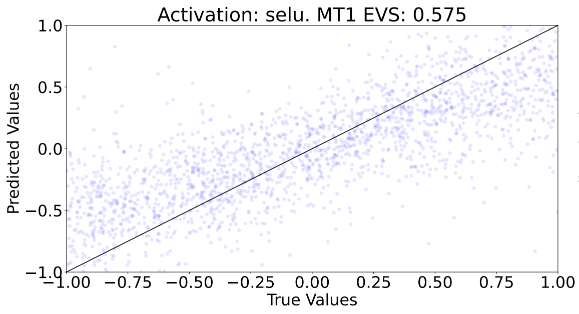
Table 4.6: Network architecture of the net used to evaluate the performance of the algorithm with changing activation funcion. Results can be found in Figure 4.10.



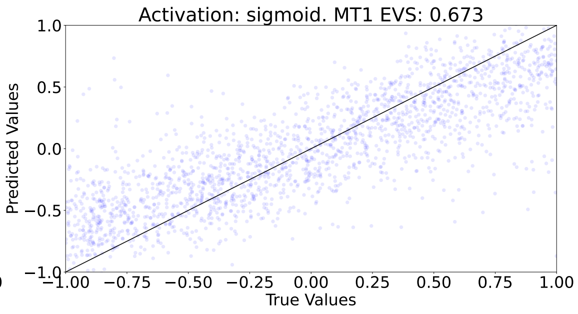
(a) Relu Activation



(b) Elu Activation



(c) Selu Activation

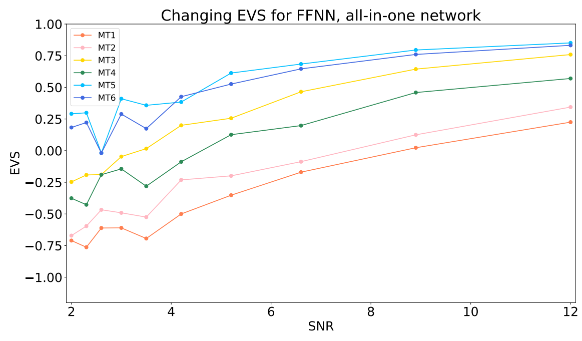


(d) Sigmoid Activation

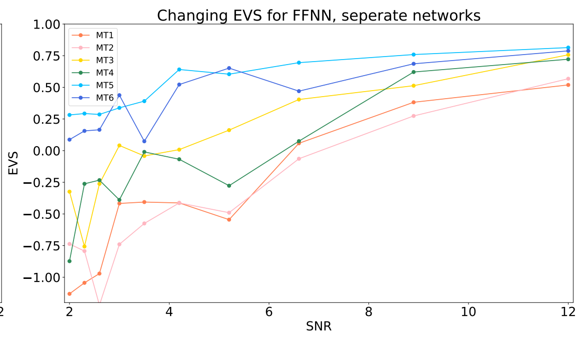
Figure 4.10: Influence of the activation function on the performance of the network whilst predicting the moment tensor components, visualized by the true versus predicted values of the test set and the explained variance score (EVS). The four plots show the explained variance score for the first moment tensors component, where each blue dot depicts a test sample. The black linear line denotes the trend a perfect predictor should have; each predicted value being exactly the same as the true value. As can be seen is that the blue dots follow this line more closely for especially the sigmoidal activation function. At the same time, the EVS for this activation function is clearly higher than for the other activation functions. The network architecture for this particular problem can be found in Table 4.6.

4.1.3 Noise

To make the data set more complex and more realistic, as a last step we have added Gaussian noise to the seismic traces that form the input to the neural network. This added Gaussian noise has a mean of zero and changing standard deviation, which is explained in SNR for each data set in the upcoming figures. From the previous section the best performing network architecture has been picked, and tested on data sets with varying SNR. Here, it was decided to check if the approach of an all-in-one network, or the individual networks made a change to the EVS. We can conclude from Figure 4.11 that this is the case. Especially for the data sets with lower SNR, we can see a more stable prediction of the individual MT components when only one network. This is, again, in contrast with what we would have expected from the literature. It might be due to the fact that the type of network we are using here is different, and that the FFNN does not predict better with separate networks.



(a) All-in-One Network



(b) Separate Networks

Figure 4.11: Influence of the noise on the performance of the network whilst predicting the moment tensor components. This figure shows the relation between the mean of the SNR of the heterogeneous data set (horizontal axis), with the EVS (vertical axis). It is seen that the the all-in-one network predicts the moment tensor components better than when the MT components would have been estimated separately. The network architecture for this particular problem can be found in Table 4.6.

4.2 Mixture Density Network

Now that we have seen how the network's performance changes due to the use of a more complex model, we can asses the uncertainty that is made with each prediction. In this chapter selected results of the different configuration settings will be shown. As was described in Section 3.2.3, is that the mixture density network gives us a mixture of Gaussians as an output, together forming the probability density function of which the network chooses the best possible answer. If we choose the mixture of Gaussians to be sixty (which, after careful selection was found to work best for this thesis), then, for each sample that we feed to the network, we obtain 60*6 means, variances, and weights. Since we estimated all six individual moment tensor components with a single network, we obtain for each output dimension sixty different means and variances, thus 360 means and 360 variances in total. This is in contrast with the amount of weights the network gives us, which totals to a number of sixty. The reason for this is, again, that we are estimating all moment tensor components with a single network. Thus, meaning that, for example, the sixth Gaussian of each output component will get the same weight in the end, irregardless of the fact that the sixth mixture of the the first component might describe this component better than the sixth mixture of another moment tensor component. This concept is explained in Figure 4.12.
We still make use of the true versus predicted plots in combination with the explained variance score, but now, as an addition, we also assess the uncertainty made with each prediction by looking at the, by the network, estimated mean, its variances, and weights. The set up of this section will closely follow that of the previous section, however, as derived from the network, in some cases the predictions or explained variance score might turn out a bit lower or higher than it did for the feed forward network. This could be explained by the change of loss function that had to be made, in order to set up the mixture density network the right way (Section 3.2.3). The type of loss function used to train the network can improve its performance, but, since changing this hyperparameter only applies to the feed forward neural networks, it was decided to not include it in this thesis. Another reason is that each network updates its weights, biases, and activations in a different way, which is beyond the control of the user. This way no network will ever be exactly the same.

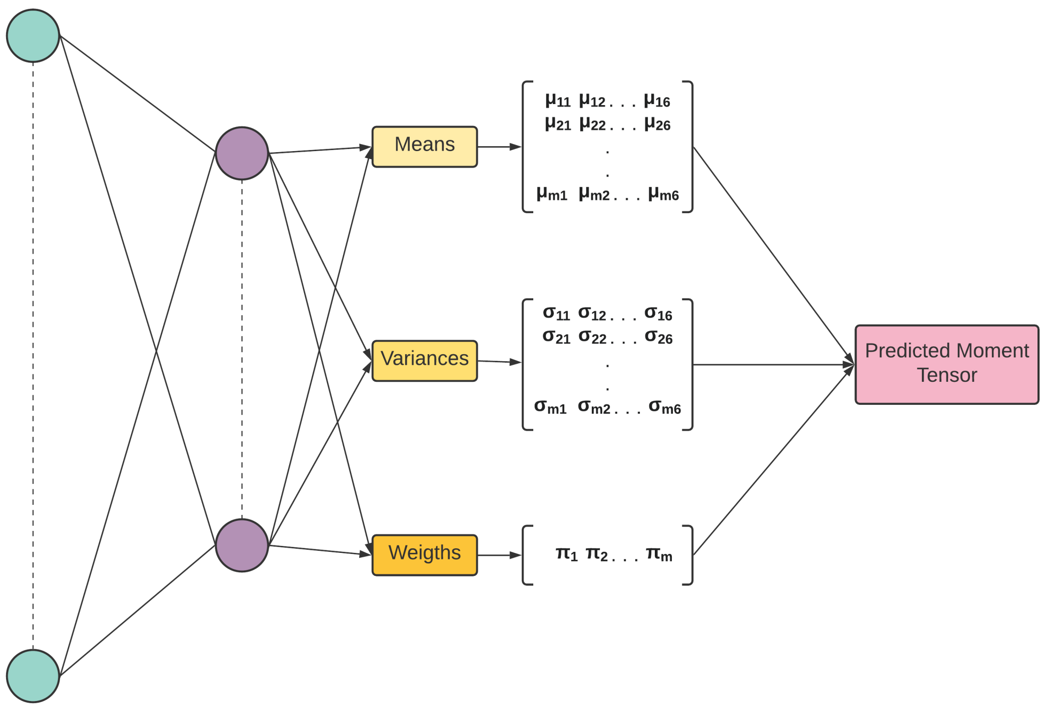
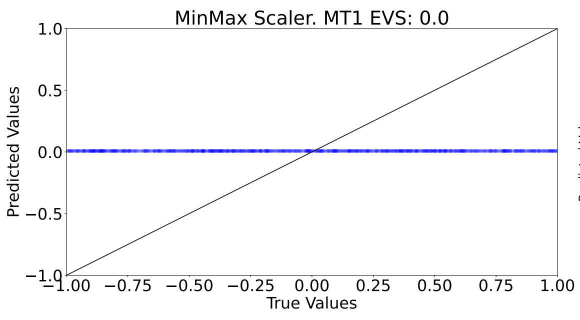


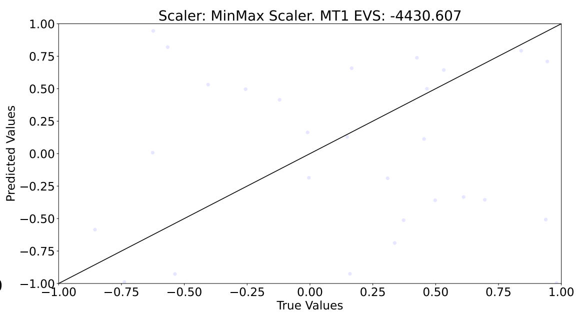
Figure 4.12: If a sample is fed to a trained network, it will be transformed by the neurons in the hidden layer to give a good estimation of the moment tensor. For a mixture density network, the output is a bit different; instead of the moment tensor components, we obtain the means, variances, and weights that correspond to each mixture that tries to describe the the moment tensor components. In this Figure we can see the output we will obtain if we feed only one sample to the network. For the means and the variances this means we will retrieve six (as in six moment tensor components) times m (the amount of mixtures) individual means and variances, and for the weights we will get a total of m weights.

4.2.1 Homogeneous Model

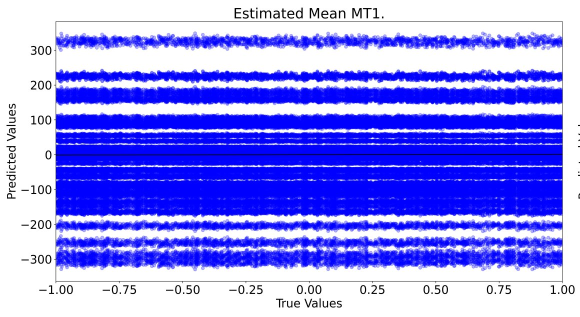
From Section 4.1.1 our results suggest that a change to either a hyperparameter or strategy does not change a lot to the performance of the network itself in the case of the simplest model, i.e., the homogeneous model. The only exceptions are those for the type of input (4.2) or output scaler (Figure 4.3) applied to the data. To give a better understanding as to what the difference between the FFNN and MDN is exactly, we will look at the uncertainties of a poor performing network with one of a good performing network. This is done by looking at the output that the network gives us, before we pick the best possible predictions, namely the mean, variance, and corresponding weight of each sample and Gaussian. Figure 4.13a shows the prediction of the network and the EVS of the same architecture as shown in Figure 4.2b, but in addition we can now also investigate the uncertainty the model has when predicting the moment tensor values. First, let us look only at the means we obtain from the trained network. As is seen in Figure 4.13b it looks like the network using the mixture density architecture will give a different sort of output than that of a feed forward neural network (Figure 4.13a). This is confirmed as well in Figure 4.13c. The estimated means are in a really big range, and applying the weights to these means does nothing to reduce this range (Figure 4.13d).



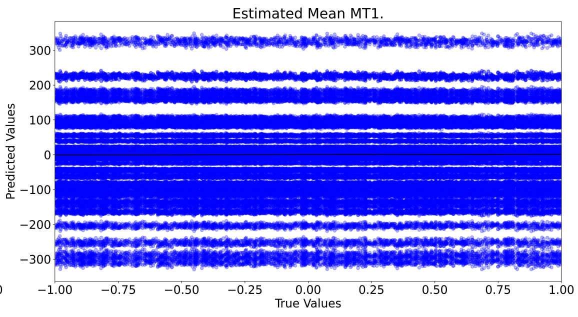
(a) Result FFNN



(b) Result MDN



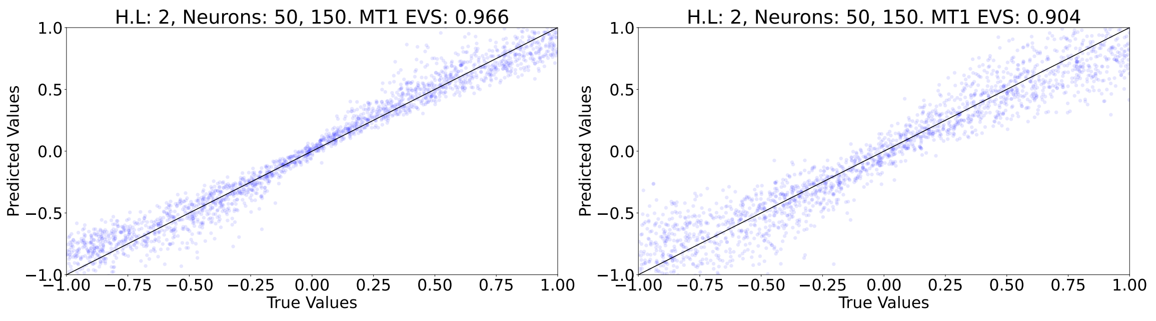
(c) Mean



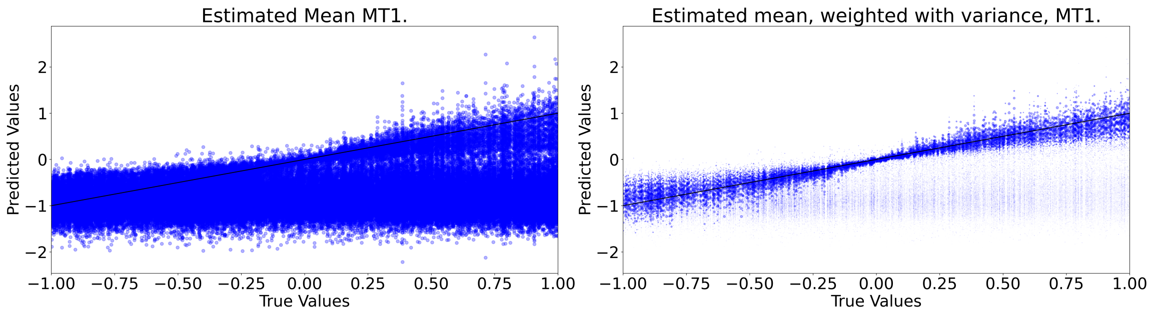
(d) Mean with weigths and variances

Figure 4.13: The performance of a network when using a data set of 50.000 earthquakes to train the algorithm. Figure 4.13a shows the performance measured by visualizing the true versus predicted values of the test set and the explained variance score (EVS) of the feed forward neural network as shown in Figure 4.2b. Figure 4.13b shows the same assessment of the network, but then from the predictions of the mixture density network. Each sample fed to the network will produce, for the amount of mixtures defined, a mean, variance and weight. Figures 4.13c and 4.13d show how the original output (means, weights, and variances) influence the accuracy of the prediction, with the former having no weights applied to each mixture, and the latter having the weights and variances applied. In these two figures, the black linear line depicts the perfect case in which the network estimates each component precisely. The network architecture for this particular problem can be found in Table 4.3.

On the other hand, Figure 4.14 shows the performance of the network with the same architecture as Figure 4.4c. The results for the EVS and the prediction of the first moment tensor components are pretty much equal. If we now look at the estimated means that the network gives us for each sample (Figure 4.14c), it seems like it is not in accordance with Figure 4.13b at all; there is a pretty wide range and we cannot really distinguish a trend around the black linear line in the figure. This is also not resolved when we apply the weighted variances to each estimation of the mean. Figure 4.13d still shows the same uncertainties for predictions.



(a) Result FFNN (b) Result MDN



(c) Mean (d) Mean with weigthed variances

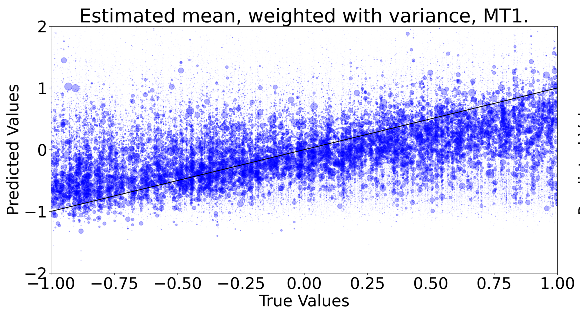
Figure 4.14: The performance of a network when using a data set of 10.000 earthquakes to train the algorithm. Figure 4.14a shows the performance measured by visualizing the true versus predicted values of the test set and the explained variance score (EVS) of the feed forward neural network as shown in Figure 4.4c. Figure 4.14b shows the same assessment of the network, but then from the predictions of the mixture density network. Each sample fed to the network will produce, for the amount of mixtures defined, a mean, variance and weight. Figures 4.14c and 4.14d show how the original output (means, weights, and variances) influence the accuracy of the prediction, with the former having no weights applied to each mixture, and the latter having the weights and variances applied. In these two figures, the black linear line depicts the perfect case in which the network estimates each component precisely. The network architecture for this particular problem can be found in Table 4.2.

4.2.2 Heterogeneous Model

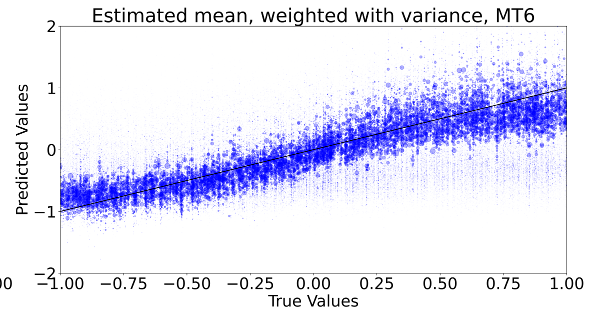
As we knew from the homogeneous model using the feed forward architecture, the model predicts well irregardless of the hyperparameter settings or strategies used (except for the scalers on input and output data). It is much more interesting to look at the heterogeneous model, since it is here that we saw that the network became more uncertain for the simpler case. The maximim amplitude scaler is used for every result shown in this section, because due to the increased complexity of the data, using no scaler is not giving good results anymore. Let us look first at the different activation functions. As was seen in Figure 4.10d, the sigmoid performed best out of the four proposed activation functions. Table 4.7 shows the network architecture of the comparison of the different activation functions made in Figure 4.15. Figure 4.15 itself shows the weighted means and variances for the first and sixth moment tensor components, firstly, to show a bit more in depth analysis of the accuracy of the network, but, also, to highlight, again, the differences in uncertainties the network shows us when predicting each moment tensor component. Next to the visible sigmoidal shape of the predicted values seen for MT6, it is interesting to note that amongst the four types of activation functions there are not that big differences in precision of the estimated moment tensor components anymore. Notable is the estimation of MT1 using a sigmoidal activation compared to the other activations. Figure 4.15g shows an almost converging value of the predicted values giving any type of input. This is in contrast with the estimation of MT6 (Figure 4.15h), where a nice trend along the linear reference line is seen.

Hyperparameter / Strategy	**Value / Type**
Number of hidden layers	2
Number of neurons per layer	50, 200
Activation Function	varying
Learning Rate	0.001
Batch Size	750
Regularization	Dropout (0.05)
Initialization	-
Scaler	Input divided by maximum amplitude
	Target values scaled from [0,2pi]
Optimization	ADAM
Size data set	10.000

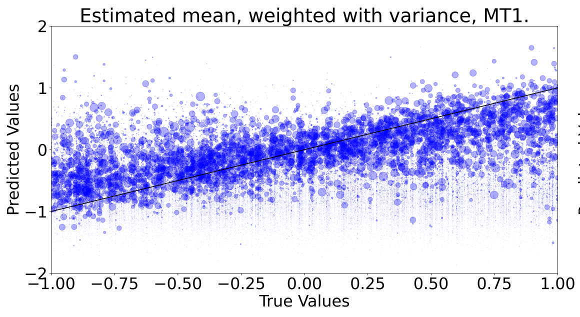
Table 4.7: Network architecture of the net used to evaluate the performance of the algorithm with changing activation function. Results can be found in Figure 4.15.



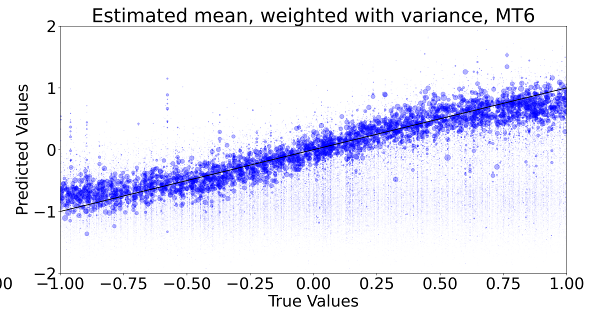
(a) Relu Activation MT1



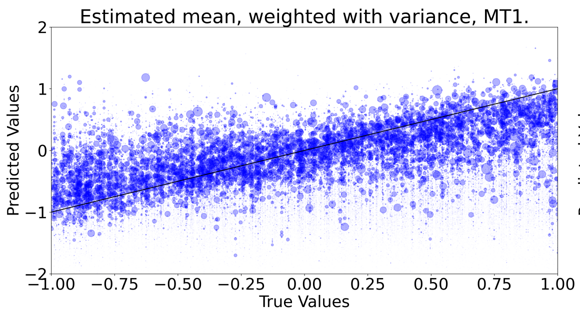
(b) Relu Activation MT6



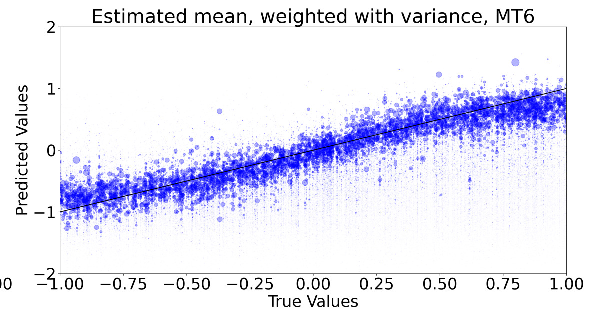
(c) Elu Activation MT1



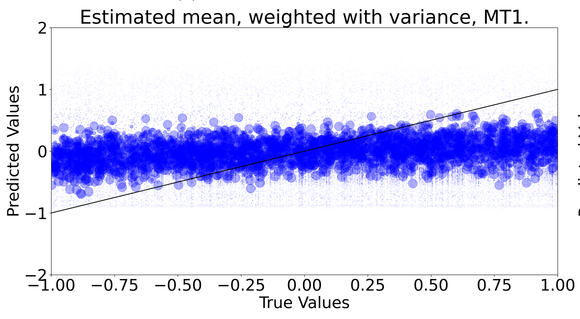
(d) Elu Activation MT6



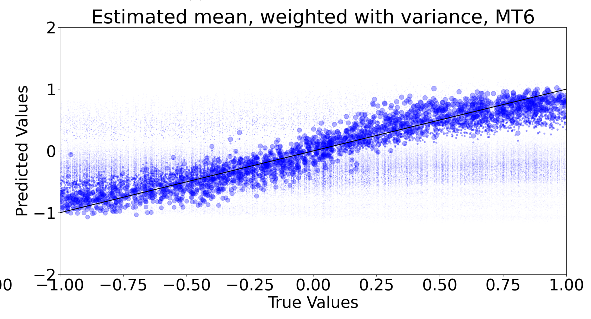
(e) Selu Activation MT1



(f) Selu Activation MT6



(g) Sigmoid Activation MT1



(h) Sigmoid Activation MT6

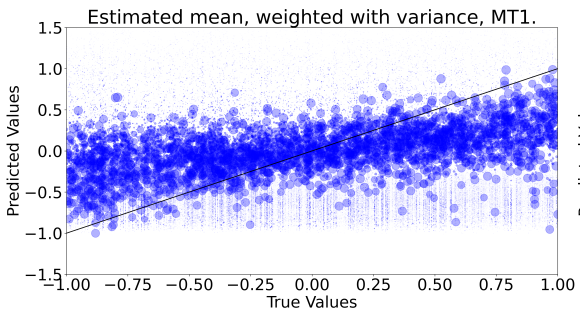
Figure 4.15: The network architecture for this particular problem can be found in Table 4.7.

Section 4.1.2 hinted at the possibility of a network performing better when only the individual moment tensor components where estimated by the network. It was later seen that this assumption did not held in the case of a feed forward network. We can try and see if this changes for the mixture density network. As (Käufl 2015) states in his dissertation, estimating each separate moment tensor component with a different individual network, should in the end yield better results. Figure 4.16 shows the comparison of four separate networks with the same architecture estimating four different moment tensor components, and the results of one network, with the same architecture as the four individual algorithms, estimating the weighted means of the four moment tensor components at the same time.
One of the first things that stands out is the shape of the sigmoidal activation function we can recognize, especially in Figure 4.16h. Unfortunately it is also this activation function that, especially when using the individual networks, limit the range of our predictions of which Figure 4.16b and Figure 4.16f are good examples. In these

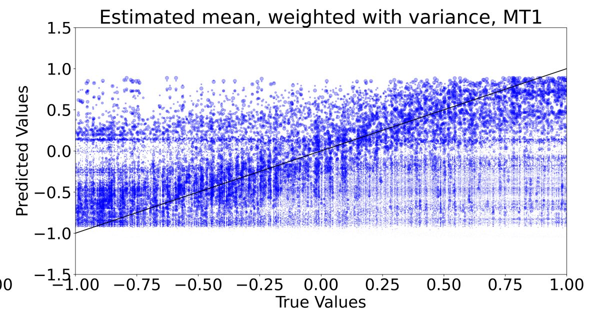
figures it is seen that values just under the range of the true moment tensor components are cut off. A reason for this anomaly was not directly found. In fact, the architecture for the two types of network is the same in each figure, but repeated six times for each moment tensor component in the case of the right side of Figure 4.16. To really see how this cut-off, especially at the upper bound of the positive values looks like without the weights applied see Figure 4.17.

Hyperparameter / Strategy	**Value / Type**
Number of hidden layers	2
Number of neurons per layer	50, 200
Activation Function	Sigmoid
Learning Rate	0.0005
Batch Size	1500
Regularization	Dropout (0.2)
Initialization	-
Scaler	Input divided by maximum amplitude
	Target values scaled from [0,2pi]
Optimization	ADAM
Size data set	10.000

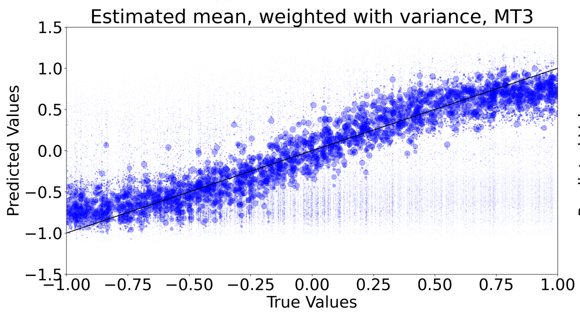
Table 4.8: Network architecture of the net used to evaluate the performance of the algorithm estimating the MT components individually and all-in-one. Results can be found in Figure 4.16.



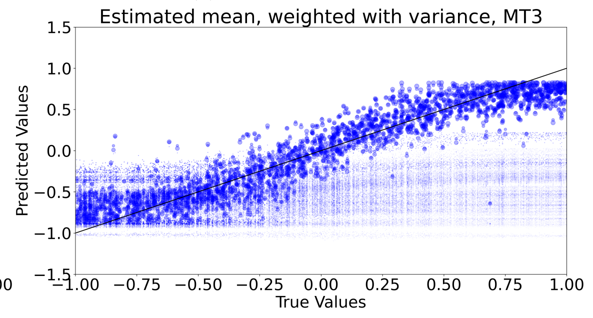
(a) One Network



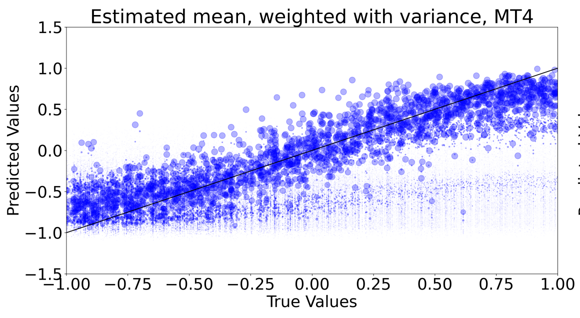
(b) Individual Network



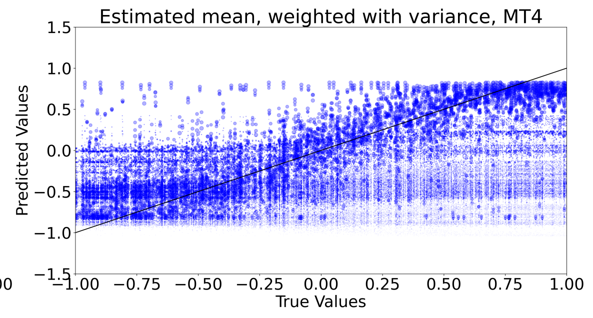
(c) One Network



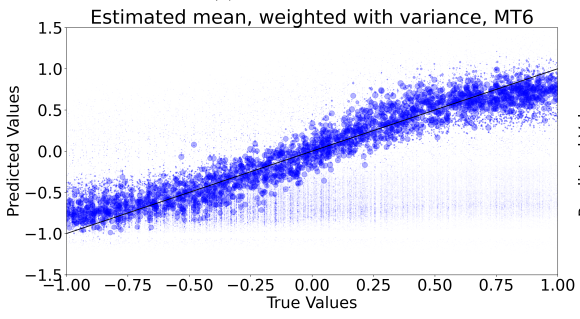
(d) Individual Network



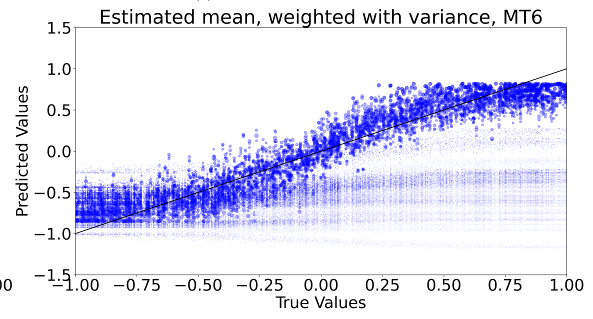
(e) One Network



(f) Individual Network

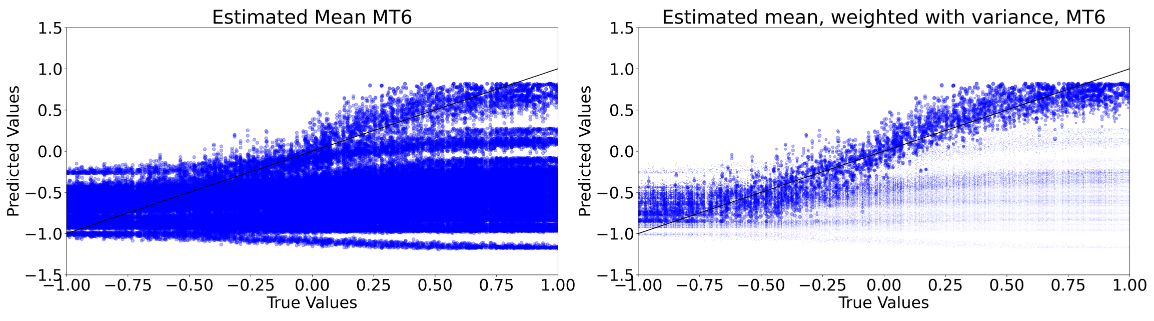


(g) One Network



(h) Individual Network

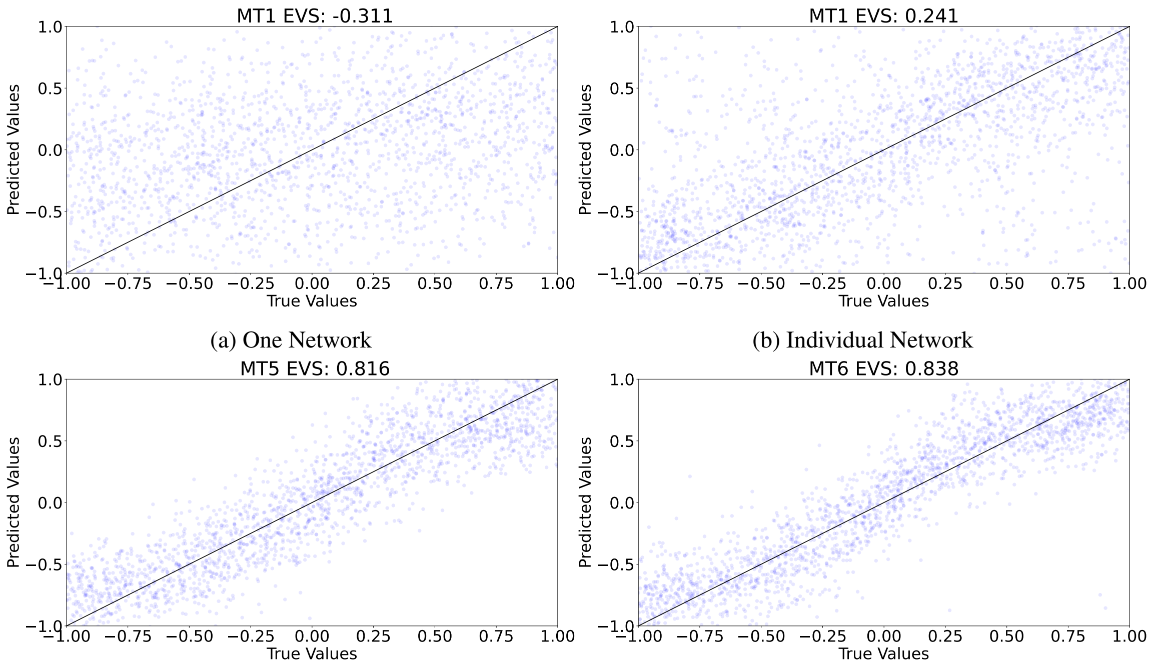
Figure 4.16: Comparison of the weighted estimated means using an all-in-one- or individual network approach for the estimation of the moment tensor components. The figures on the left side show the estimated, weighted means for MT1, MT3, MT4, and MT6 from one single network (the all-in-one network). We can see that the estimated means are pretty certain, and that they follow the black reference line. The figures on the right show the estimated, weighted means for MT1, MT3, MT4, and MT6, where each MT component was estimated by its own network. Here, all the individual algorithms have the same network as the all-in-one network. It is seen that for some components the network is less sure of its own prediction (for example MT1 and MT4). Next to this, the shape of the sigmoid activation function used for this particular problem is clearly seen in the figures on the right side. The network architecture for this particular problem can be found in Table 4.8.



(a) No weights applied (b) Weights applied

Figure 4.17: Comparing the result of Figure 4.16h, where the weights are applied to the estimated means of the network estimating only MT6, to the same estimated means without the weights applied. It is seen that there is a cut-off of the predicted values at the upper limit, whereas the lower limit seems to form a linear dipping boundary.

Overall we could say the the uncertainty of the networks prediction is less when estimating all moment tensor components using only one network, however the estimation of the first moment tensor component could be an exception. In Figure 4.16a there is a linear trend visible, however comparing this trend in Figure 4.16b one could argue that the latter might be a bit better at predicting the MT components.
To verify this, let us look at the actual predicted values that we draw from the distributions obtained and showed in Figure 4.18. Figure 4.18 shows the EVS for the first moment tensor component of the all-in-one network (Figure 4.18a) and of the individual network (Figure 4.18b). From this Figure we can conclude that indeed, MT1 is better estimated by the individual network in terms of the explained variance score, and, also from the true versus predicted plot, we see a better estimation for the individual network. This is in contrast with Figures 4.18c and 4.18d, where the individual network only performs slightly better. Keeping in mind that using different network architectures for estimating the individual MT components in Section 4.1.2 also did not improve the EVS that much, we can ask ourselves why we should even look into individualizing the estimation of the different MT components. In the next Section, we will see that adding Gaussian noise to our data in combination with the separate networks makes the network perform better for lower SNR.



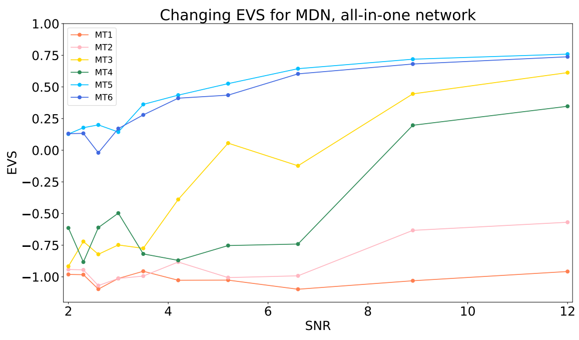
(a) One Network (b) Individual Network

(c) One Network (d) Individual Network

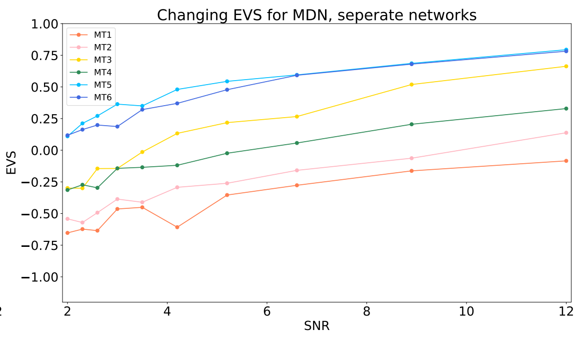
Figure 4.18: Comparison of the EVS for MT1 and MT6 using an individual- or all-in-one network. The first moment tensor component benefits more from the individual network in terms of EVS and precision than the sixth moment tensor component does. Figure 4.18c and 4.18d do not seem to differ that much from each other.

4.2.3 Noise

In contrast to Section 4.1.3, using a MDN we see that the network performs better with lower SNR if seperate networks are used to estimate the MT components (Figure 4.19. This is also in line with (Käufl 2015). Another advantage of using separate networks, is that we can construct the joint PDFs to evaluate the trade-off of the individual moment tensor components.



(a) All-in-One Network



(b) Separate Networks

Figure 4.19: Influence of the noise on the performance of the network whilst predicting the moment tensor components, The network architecture for this particular problem can be found in Table 4.6.

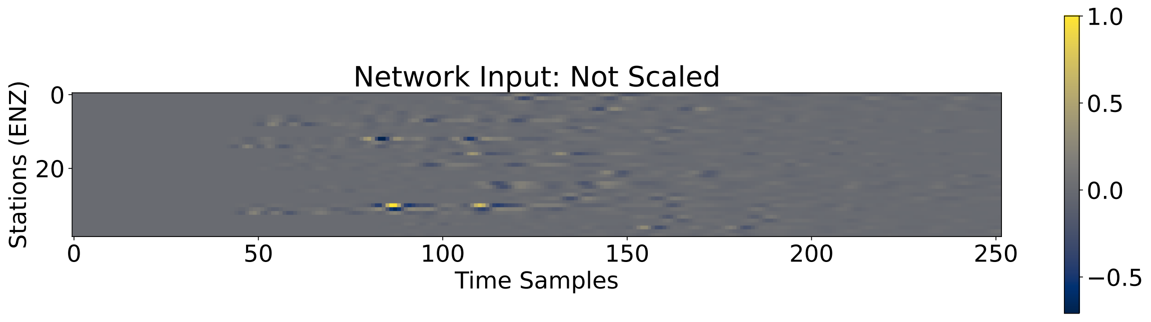
Chapter 5

Discussion

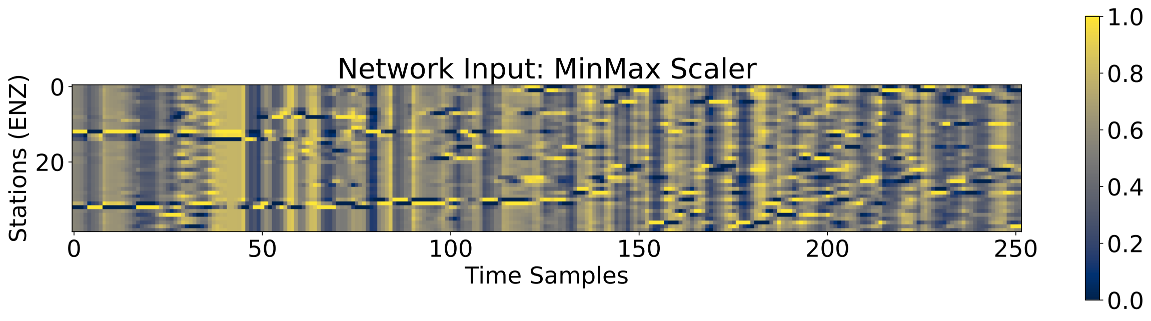
There are a few things that should be addressed in this section. The order of the outcomes discussed is loosely following the occurrence of these results in the previous section. As an extra point of discussion we also show how the amount of data influences the performance of the network at the end of this section.

Scalers

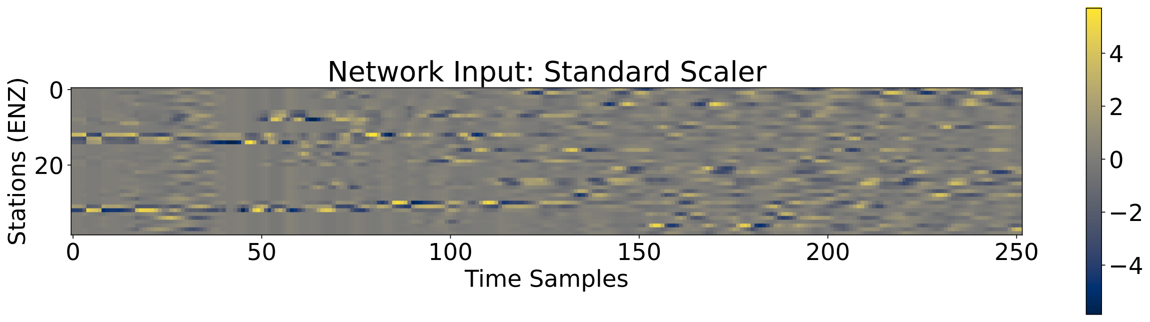
Figure 4.2 shows how four different scalers yield different results with respect to the performance of the network. It was seen that, only for the homogeneous model, no big differences were found between using no scaler or a scaler that scales by the maximum amplitude in an event. This is an interesting result, since not scaling the input data could lead to an unstable learning process, or even convergence to a certain mean. That this does not happen for the case were the network is trained on samples that are modelled using a homogeneous velocity model is striking. An explanation for this could be that since the magnitude of all events are fixed and the model is not complex, there are no big outliers in the data set, which means that a scaler does not necessarily needs to be applied. However, for a more complex model (and especially when a range of magnitudes is considered in future work), a scaler does need to be applied.
The two more commonly used scalers for neural networks, namely the MinMax- and Standard scaler, result into a very poor performing network for our case. The reason why these scalers perform so poorly is quickly found if we look at how exactly the scalers transform the input to the network. Figure 5.1 shows the input of one event unscaled, and the transformation it has undergone after being scaled by the three scalers investigated in this thesis.



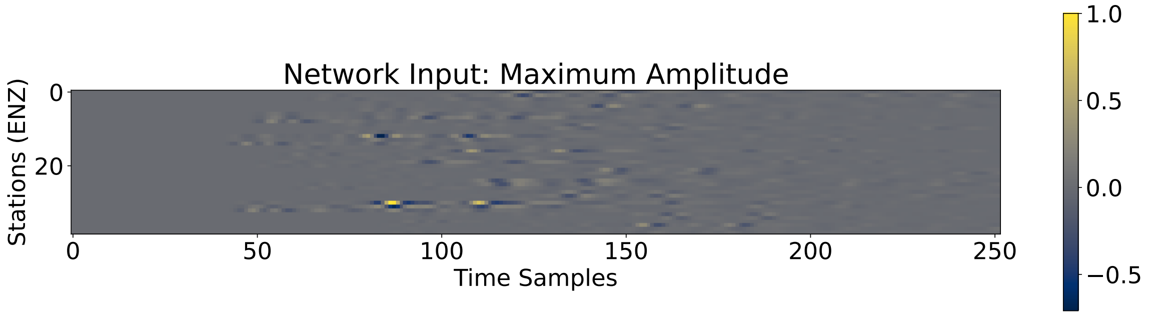
(a) No Scaler



(b) MinMax Scaler



(c) Mean



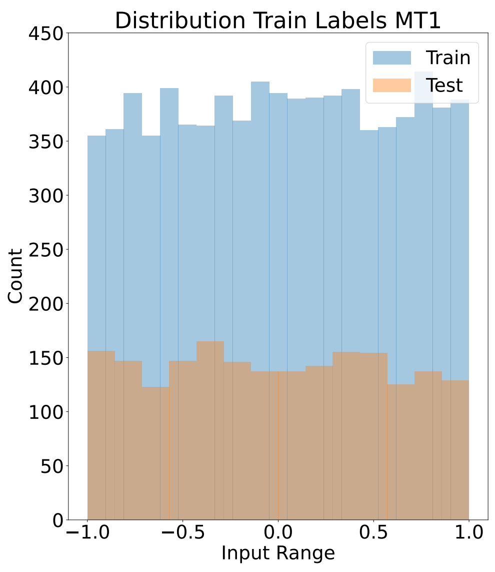
(d) Maximum Amplitude Scaler

Figure 5.1: Part 2: What effect does a scaler have on the input data? In these four figures we can see the unscaled input of one event (Figure 5.1a), and the scaled input after using the MinMax scaler (Figure 5.1b), the Standard scaler (Figure 5.1c), and after scaling by the biggest amplitude in the event (Figure 5.1d). On the horizontal axis we see the time samples, where each time sample has an interval of 0.04 seconds. On the vertical axis the ENZ component data of each of the 13 stations is plotted underneath each other.

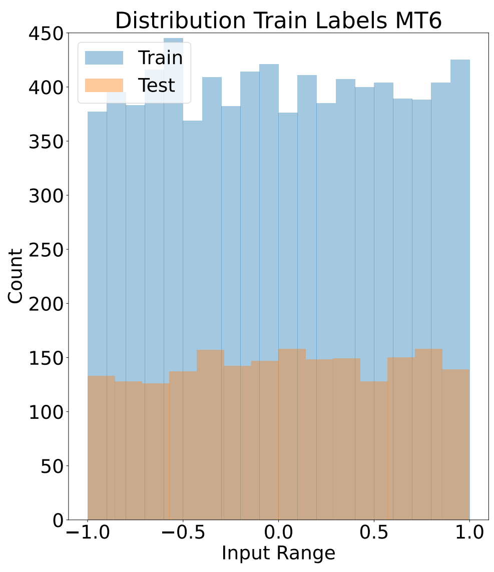
Figure 5.1a and 5.1d seem to be identical, however, this is mostly due to the amplitude information being kept. The MinMax scaler (Figure 5.1b) clearly shows that something is going wrong while scaling the data. We think that the scaler scales the output per time sample, instead of globally. The linear lines that appear in the plot must be present already in the synthetic data, possibly with very small amplitudes, which normally will not be seen. It is due to the extreme scaling that boosts these nearly zero amplitudes that the vertical lines become more pronounced. Figure 5.1c shows the output of the standard scaler. It does not look as bad as Figure 5.1b, however, we can see that some of the three component data shows oscillation in the first time samples, and that the data from the later time samples is really emphasized by this scaler. This is undesirable, since we expect to get the most information on the moment tensor components from the first time samples.
By scaling each event by its maximum amplitude we will remove information on the magnitude of each earthquake. This could form a problem when the difference in SNR is big between the individual events.

Sensitivity of the MT components with complex data

Normally, when there is a difference in sensitivity of the network to different output neurons, we would say that the output neuron with the lowest sensitivity, i.e., the highest errors, is underrepresented in the data set. In our case this is not true, and we can show this by plotting the distribution of the MT1 and MT6 components of the train and test set (Figure 5.2). Apart from that our results showed that only when switching to a more complex data set a shift in the sensitivity of the network to some of the MT components was seen. Generally the off-diagonal components are still well resolved, but the diagonal components, especially MT1 and MT2, show a performance that is a lot poorer than with a less complex model. An explanation for this could be that these diagonal components represent volumetric changes, or, the ISO component. With a pure isotropic source, the amplitudes in the time traces will be the same for every azimuth and take-off angle. For a homogeneous model this uniformly distributed amplitude is nicely picked out by the network. When switching to a more complex data set, we are adding an extra layer of difficulty via the Green's Functions that correspond to the heterogeneous model. It seems that the combination of this addition, the different take-off angles, and azimuths makes it harder for the network to accurately predict the isotropic components.



(a) MT1



(b) MT6

Figure 5.2: Distribution of train and test labels for MT1 and MT6

Activation Function

Training the algorithm with a FFNN shows that the sigmoidal activation function works best. Considering that this particular function often causes vanishing gradients, it should be unusual that it works in the case of moment tensor estimation. One reason why it could work, is because the maximum amplitudes for each event themselves do not differ that much from each other, due to the fixed magnitude and the small area of investigation. By dividing every event by almost the same maximum amplitude only very small differences between the data samples could occur. The vanishing gradient is caused by the small range in which the sigmoid function transforms the output of the neuron, causing the gradient to become very small, which in turns leads to the model not updating the neurons anymore. This most often happens when a data set contains a lot of different data. This is not the case in our data set, which could explain why the sigmoidal activation function works well. Interestingly, it is also the sigmoidal function that is often used for probability estimation. Yet, when we look at the performance of the sigmoid activation function in combination with the MDN, it is seen that this particular function does not yield the best performance anymore, especially with regard to MT1. A better recommendation for the MDN network could be, for example, the Selu activation function.

Since we know that the diagonal components do not cause a lot of changes to the traces, and the range of MT1 components is uniformly distributed, it is not very probable that we are dealing with a vanishing gradient due to the sigmoid activation function, even though the converging of the predicted values might indicate otherwise.

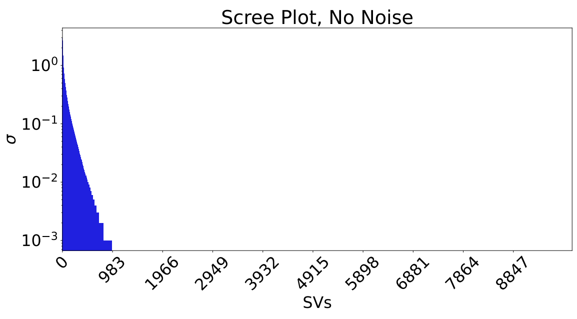
FFNN or MDN

Following the results, there are not big differences between the approach of using a FFNN or MDN. They both roughly predict the moment tensor with the same precision, and in terms of computational cost there is also not a lot of difference. From a perspective of wanting to know the uncertainty of the network's prediction, a MDN would definitely be the better option. That leaves us with the question of estimating all MT components in one network or predicting them separately. For a FFNN it showed that predicting the MT components separately actually does not improve, or maybe even worsen in the case of a model with noise, the precision of the predicted moment tensor. From the first tests with separate and all-in-one networks for the MDN we could draw the same conclusion, yet when we switched to a more complex model, i.e., with noise, this changed. Here, it was seen that the network was more stable when dealing with data containing a lower SNR. If this thesis is to be expanded to real data from the Zeerijp area, which often contains lower SNR due to the micro earthquakes, separately estimating the MT components would definitely be of more interest.

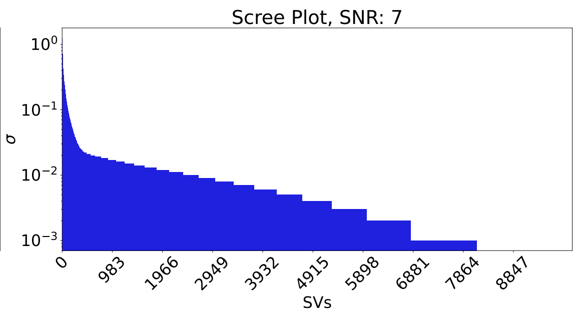
Amount of Data

How much data is needed to accurately train the network? The choice of training with 10.000 earthquakes sounds arbitrary, and in some ways it is. This amount of data is mainly based on the fact that (Binder 2018) found that with a train set of 10.000 earthquakes, the network trains nicely, but it also has to do with the storage space available on the server, and the computational effort (it takes around 90 CPU hours to generate this data set). To investigate if less data could be used for future research it was decided to perform a singular value decomposition (SVD) on the features in the heterogeneous data set that contains no noise. Here, we assume that the amount of non-zero singular values corresponds to the amount of data we theoretically would need to obtain a good performance of the network. We make this assumption as follows. The matrix with the singular values on the diagonal will have the size of the smallest dimension, in this case it is the input to our network, namely thirteen stations times three component data multiplied by the number of time samples (251), which totals to 9789 values. If we find the case where the amount of non-zero singular values is equal to the size of the diagonal of this matrix, it would mean that every event in the data set represents an unique feature. Of course, one could argue that also the amount of stations recording the event and the length of the time traces play a role here, which is true, however this could be a whole other research on its own. To elaborate on this; changing the stations or length of the time trace alters the scree plot. Changing the data size does not change the singular value itself, only the dimension of the singular values will differ. In other words, the amount of singular values is cut off at the dimension of the data set chosen.

We can therefore say that, with the amount of stations and the length of the time window fixed, if we look at Figure 5.3a it is clearly seen that the network could be trained with less data. If we execute the same decomposition for the data set containing noise, we can see that in fact the estimate of 10.000 earthquakes is very accurate. From Figure 5.3a we could deduce that a no noise, heterogeneous network could be trained with less than 1000 samples. If we add noise to this same data set (which will account for a SNR of roughly 7 over the complete data set) the amount of non-zero singular values increases to a mere 8000, see Figure 5.3b. This means that in order for the network to converge on the former data set, we need at least 8000 training samples.



(a) Heterogeneous model, no noise



(b) Heterogeneous model, with noise

Figure 5.3: Scree plot showing the singular values for the heterogeneous data set containing no noise (left), and the same data set containing noise (right). It is seen that with the added complexity of noise to the model, the amount of non-zero singular values increases.

Chapter 6
Conclusion

What architecture and which type of neural network can predict the moment tensor of the earthquakes in the Zeerijp region of the Groningen Gas Field area? And how can we assess the precision of these results?

To answer the first question: both the FFNN and MDN type of neural networks can predict the moment tensor, and there are several combinations of hyperparameters in the network architecture that can achieve this. This makes finding the right network to obtain the solution to the moment tensor a non-unique problem; there is no one right way. The main reason for comparing the FFNN and MDN was to see if there really is that much difference in precision and computational cost. From our results we can conclude that yes, the two networks need different hyperparameters and strategies in order to train well, but in terms of computational cost there is actually not that much difference. In fact, the computational costs, at its most, round up to around 100 CPU hours for a more complex model. Of course, using, for example, more data, would contribute to longer training times, but overall the network trains fast.

The second question was answered by considering data with different complexity. We can conclude that the more complex the data set is, the more uncertainty the network experiences predicting the moment tensor, in particular for MT1 and MT2 (they represent volumetric changes, which are harder to discern in the input data). That the estimation of the MT becomes more uncertain with a complexer data was what we expected, the differences in sensitivities not. The uncertainty of each network was assessed by the use of the EVS score, and the output of the MDN network. Predicting the moment tensor components individually shows a slight increase in the precision of the estimations (for the heterogeneous MDN case, this is not necessarily the case for the FFNN) and an overall more stable network for lower SNR, however, if we look at the weighted estimates means (so the uncertainty of the network) it still shows that not each MT component can be predicted with the same precision.

A final comparison of one of the better performing networks using either the FFNN or MDN type of network is shown in Figure 6.1. In this Figure only the data sets without noise are shown, where all components are estimated by one network. From this plot it is clearly seen that the FFNN, in terms of EVS, will always give a better estimation of the MT, irregardless of the type of data set used. However, knowing that the MDN also provides us with the related uncertainties of these predictions, and gives a more stable prediction when adding Gaussian noise to the data, it might be better to continue future research with this type of data.

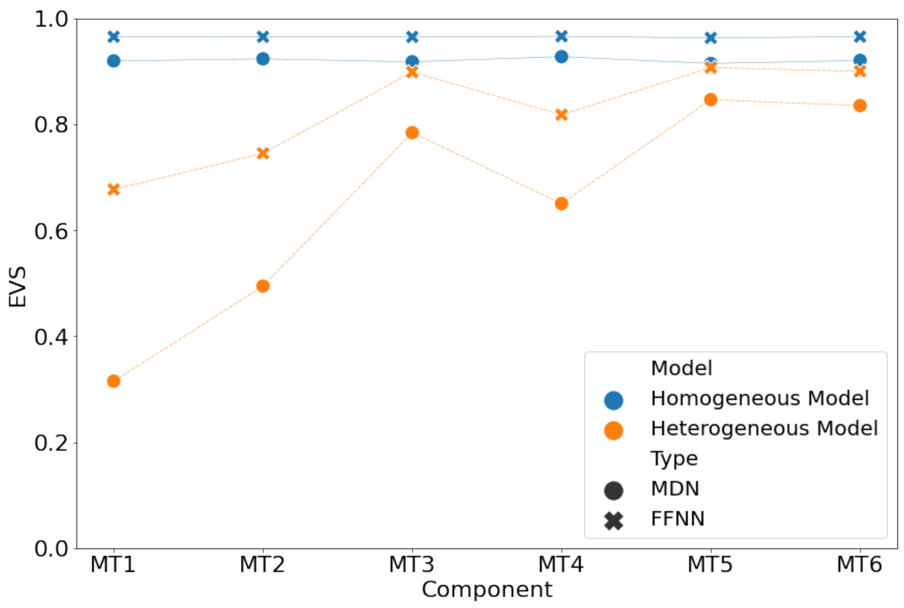


Figure 6.1: Comparison of the best performing MDN and FFNN for the homogeneous- and heterogeneous data set containing no noise.

Chapter 7

Outlook

The results obtained in this thesis indicate that it should be very possible to build a neural network that can estimate the moment tensor components of the earthquakes taking place in the Zeerijp area located in the Groningen Gas Field. For future research there are a few recommendations

1. **Vary the magnitude** in the data set; the networks trained in this thesis were all focused on earthquakes with a fixed magnitude. From our results the scaler that transforms the data by the maximum amplitude measured in the response of an event works best. One of the side effects of this scaler is that all magnitude info between the events are lost. It might be that a data set with varying magnitude will not have a big effect on the precision of the network, but it is worth looking into, since it will make the training set more realistic.
2. **Add measured noise** to the data set. Noise in the Zeerijp area is cyclic. An advantage of using this noise instead of Gaussian noise while training could be that the network learns the cyclic noise, thus picking up smaller changes in amplitude caused by micro earthquakes. These micro events form the vast majority of the earthquakes happening in the area, thus it is important for the network to also recognize the moment tensor from the corresponding traces.
3. Analyze the most common fault mechanism in the Zeerijp area to **bias the data set** more towards the region of interest. The data set used in this thesis is biased towards the Zeerijp area in the sense that it contains the actual receiver and fault locations in the region, but the moment tensors are completely random. By adding more synthetic events with a Zeerijp representative moment tensor (normal faulting), the network could perform better when fed real data.
4. Investigate separating the prediction of MT components over **several networks**; By constructing joint probability distributions between various MT components we can not only obtain more information on the uncertainty of the predictions and the trade off between the components, but we can also try more advanced network architectures for the diagonal MT components in the hope of increasing the EVS.
5. **Add more output parameters** to increase the performance of the network. Including to the output of the network the strike, dip, and rake during training, we think that the low sensitivity for the diagonal components could be compensated for, since we would have an extra means of obtaining information on the focal mechanism. Since the moment tensor already contains the information regarding the strike, dip, and rake, this might seem unnecessary to add as extra output parameters. Yet, if the network proves being able to better predict the strike, dip, and rake angles, it could complement the lower sensitivity to MT1 and MT2, thus improving the overall performance of the algorithm.
6. **Change the velocity model** to investigate the impact of the salt layer on the complexity of the time signals. It is known that anhydrite layers transforms the response of the earth and generally make it harder to discern any useful information from the trace. By using the same 1D velocity model, but leaving out the salt layers, we could see the effect of the anhydrite on the performance of the network.

Including point one to three, a network can be trained that will be robust enough to be tested on the real data from the Zeerijp area. The fifth and sixth points are mainly to improve the low sensitivity of the network to the first two MT components, which in turn could also aid the performance of the network when tested on real data.

Bibliography

Alvizuri, C. & Tape, C. (2018), 'Full moment tensor analysis of nuclear explosions in north korea', *Seismological Research Letters* **89**(6), 2139–2151.

Binder, G. (2018), 'Neural networks for moment-tensor inversion of surface microseismic data', *SEG Technical Program Expanded Abstracts* pp. 2917–2921.

Bishop, C. M. (1994), 'Mixture density networks'.

Bishop, C. M. (2006), *Pattern Recognition and Machine Learning (Information Science and Statistics)*, Springer-Verlag, Berlin, Heidelberg.

Bourne, S., Oates, S., van Elk, J. & Doornhof, D. (2014), 'A seismological model for earthquakes induced by fluid extraction from a subsurface reservoir', *Journal of Geophysical Research Solid Earth* **119**(12), 8991–9015.

Buijze, L., van den Bogert, P., Wassing, B., Orlic, B. & ten Veen, J. (2017), 'Fault reactivation mechanisms and dynamic rupture modelling of depletion-induced seismic events in a rotliegend gas reservoir', *Netherlands Journal of Geosciences* **96**(5), s131–s148.

Cox, S. C. & Allen, S. K. (2009), 'Vampire rock avalanches of january 2008 and 2003, southern alps, new zealand', *Landslides* **6**(2), 161–166.

Dahm, T. & Krüger, F. (2014), 'Moment tensor inversion and moment tensor interpretation'.

de Jager, J. & Visser, C. (2017), 'Geology of the groningen field – an overview', *Netherlands Journal of Geosciences* **96**(5), s3–s15.

Developers, T. P. (n.d.), 'Pyrocko: A versatile seismology toolkit for Python.'.
URL: *http://pyrocko.org*

Dost, B., Ruigrok, E. & Spetzler, J. (2017), 'Development of seismicity and probabilistic hazard assessment for the groningen gas field', *Netherlands Journal of Geosciences* **96**(5), s235–s245.

Dost, B., van Stiphout, A., Kühn, D., Kortekaas, M., Ruigrok, E. & Heimann, S. (2020), 'Probabilistic moment tensor inversion for hydrocarbon-induced seismicity in the groningen gas field, the netherlands, part 2: Application', *Bulletin of the Seismological Society of America* **110**, 2112–2123.

Eyre, T. S., Eaton, D. W., Zecevic, M., D'Amico, D. & Kolos, D. (2019), 'Microseismicity reveals fault activation before mw 4.1 hydraulic-fracturing induced earthquake', *Geophysical Journal International* **218**(1), 534–546.

Historiek (2019), 'Gaswinning in groningen; geschiedenis, gevolgen en toekomst', `https://historiek.net/gaswinning-in-groningen-geschiedenis-gevolgen/74692/`. Accessed: 30-06-2021.

Käufl, P. J. (2015), Rapid probabilistic source inversion using pattern recognition, PhD thesis, Utrecht University.

Klambauer, G., Unterthiner, T., Mayr, A. & Hochreiter, S. (2017), 'Self-normalizing neural networks', *CoRR* **abs/1706.02515**.
URL: *http://arxiv.org/abs/1706.02515*

KNMI (2020), 'Moment tensors – a practical guide'.
URL: *https://www.knmi.nl/kennis-en-datacentrum/uitleg/aardbevingen-door-gaswinning*

Knopoff, L. & Randall, M. (1970), 'The compensated linear-vector dipole: A possible mechanism for deep earthquakes', *Journal of Geophysical Research* **75**(26), 4957–4963.

Kraaijpoel, D. & Dost, B. (2013), 'Implications of salt-related propagation and mode conversion effects on the analysis of induced seismicity', *Journal of Seismology* **17**(1), 95–107.

Kuang, W., Yuan, C. & Zhang, J. (2021), 'Real-time determination of earthquake focal mechanism via deep learning', *Nature Communications* **12**.

Kühn, D., Heimann, S., Isken, M. P.and Ruigrok, E. & Dost, B. (2020), 'Probabilistic moment tensor inversion for hydrocarbon-induced seismicity in the groningen gas field, the netherlands, part 1: Testing', **110**(5), 2095–2111.

Romijn, R. (2017*a*), 'Groningen Velocity Model 2017 - Groningen full elastic velocity model September 2017'.
URL: *https://nam-feitenencijfers.data-app.nl/download/rapport/9a5751d9-2ff5-4b6a-9c25-e37e76976bc1?open=true*

Romijn, R. (2017*b*), 'Groningen velocity model 2017 – groningen full elastic velocity model,', *NAM Report* p. 12.

Steinberg, A., Vasyura-Bathke, H., Gaebler, P., Ohrnberger, M. & Ceranna, L. (2021), 'Estimation of seismic moment tensors using variational inference machine learning', *Earth and Space Science Open Archive* p. 22.
URL: *https://doi.org/10.1002/essoar.10506663.1*

Tarantola, A. (1984), 'Inversion of seismic reflection data in the acoustic approximation', *Geophysics* **49**, 1259–1266.

Tierney, S. (2019), 'Moment tensors – a practical guide'.
URL: *https://mxrap.com/moment-tensors-a-practical-guide/*

Vavrycuk, V. (2014), 'Moment tensor decompositions revisited', *Journal of Seismology* **19**.

Verwijs, R. (2021), 'An intro to moment tensor inversions'. Research Module part of a course at the RWTH Aachen as preperation for the Master thesis.

Vlek, C. (2018), 'Induced earthquakes from long-term gas extraction in groningen, the netherlands: Statistical analysis and prognosis for acceptable-risk regulation: Induced earthquakes from long-term gas extraction in groningen', *Risk Analysis* **38**.

Wapenaar, K., Draganov, D., Snieder, R., Campman, X. & Verdel, A. (2010), 'Tutorial on seismic interferometry: Part 1 — basic principles and applications', *GEOPHYSICS* **75**, 75A195–75A209.

Wapenaar, K. & Snieder, R. (2007), 'From order to disorder to order: A philosophical view on seismic interferometry', *EG Technical Program Expanded Abstracts* pp. 2683–2687.

Willacy, C., Dedem, E., Minisini, S., Li, J., Blokland, J., Das, I. & Droujinine, A. (2019*a*), 'Full-waveform event location and moment tensor inversion for induced seismicity', *GEOPHYSICS* **84**.

Willacy, C., Dedem, E., Minisini, S., Li, J., Blokland, J., Das, I. & Droujinine, A. (2019*b*), 'Full-waveform event location and moment tensor inversion for induced seismicity', *GEOPHYSICS* **84**.

Wilson, A., Roelofs, R., Stern, M., Srebro, N. & Recht, B. (2017), 'The marginal value of adaptive gradient methods in machine learning'.
URL: *https://arxiv.org/abs/1705.08292*

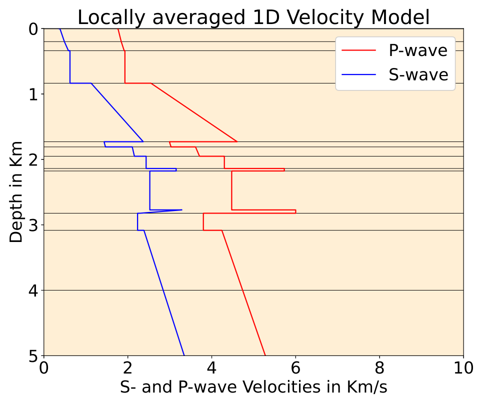
Chapter A

Trace Analysis

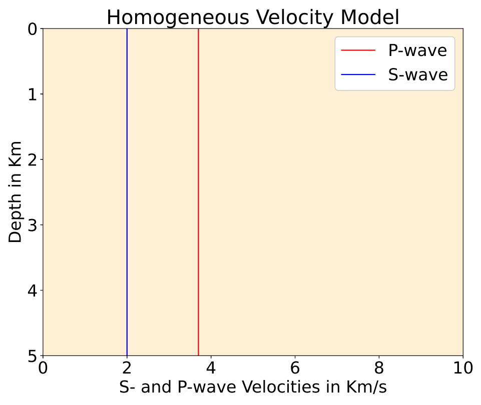
Table 3.2 showed how much a response is changed by a small alteration of one parameter for three different categories (Velocity model, source specific-, and receiver specific settings). In this chapter we will elaborate a bit more on the analysis behind this table, however still only show a selection of the results.

A.1 Velocity Model

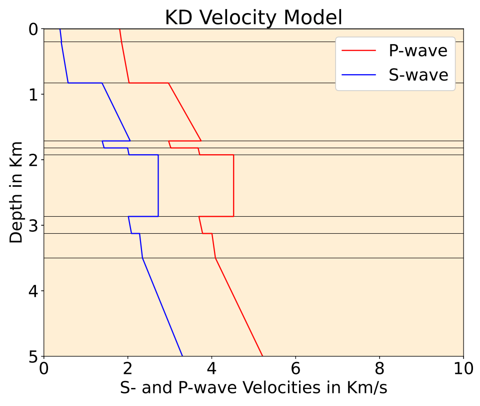
In the first category (velocity model) four different models were tested. These velocity models are used by Kühn et al. (2020) to test the performance of their inversion, where they found that the locally averaged 1D velocity model works best. Taking over this recommendation for the thesis, it is only the 1D model, and the homogeneous model that are showed in Chapter 3. In this chapter we would like to show the reader the other two velocity models as well. In Figure A.1, it is seen that the model by the KNMI (the NN model, Figure A.1d) averages layers to big blocks. The model by Kraaijpoel & Dost (2013) shows similarities to the locally averaged 1D model, but it is here that the two thin anhydrite layers are missing. The NN model could therefore be interesting to test the effect of the salt layers on the performance of the network itself.



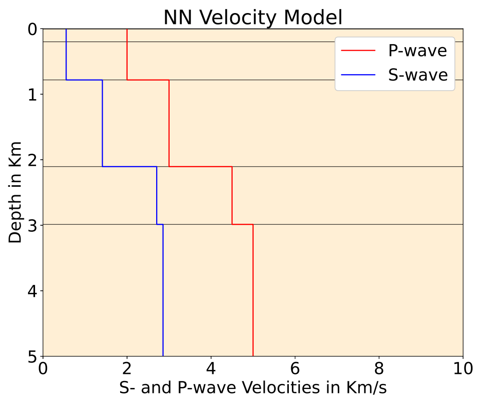
(a) Locally Averaged 1D Model



(b) Homogeneous Model



(c) KD Model



(d) NN Model

Figure A.1: Comparison of all four velocity models used by Kühn et al. (2020). If we take the 1D model as reference to the actual geology in the Zeerijp area, it is seen that especially the NN is an averaged model that very loosely follows the 1D model.

A.2 Source Specific Changes

By changing strike, dip, and rake for each of the three components we can investigate the influence they have on the signal. Strike specific changes are shown in Figures A.2 to A.4, dip in Figures A.5 to A.7, and rake Figures A.8 to A.10.

A small change in strike seems to not change the signal of the E-component a lot, whereas a positive change shows a big change for the N-component, but a negative adjustment does not. The Z-components stays pretty much the same. This is logical, since a change in strike does not necessarily affects the vertical ground motion. This change for the Z-component is seen when the dip or rake is altered. We could say that the response of the earth is least sensitive to a change in strike, and more so to the changes in dip and rake.

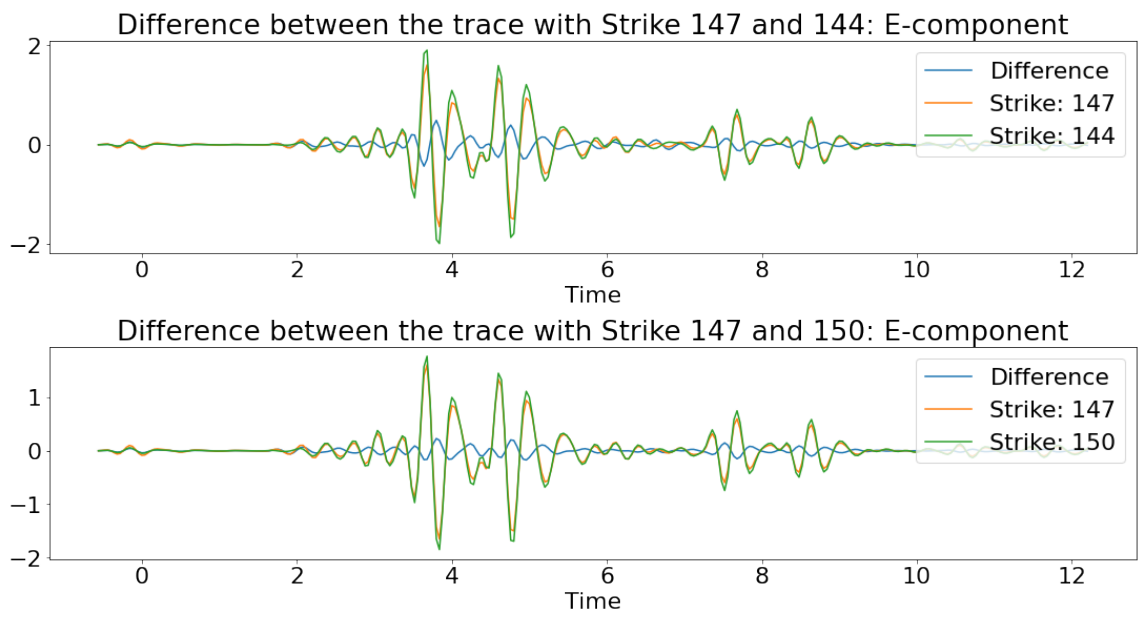


Figure A.2: Difference between two traces(green and orange), where the only difference is a small positive or negative change in the strike. The change is shown with by blue line.

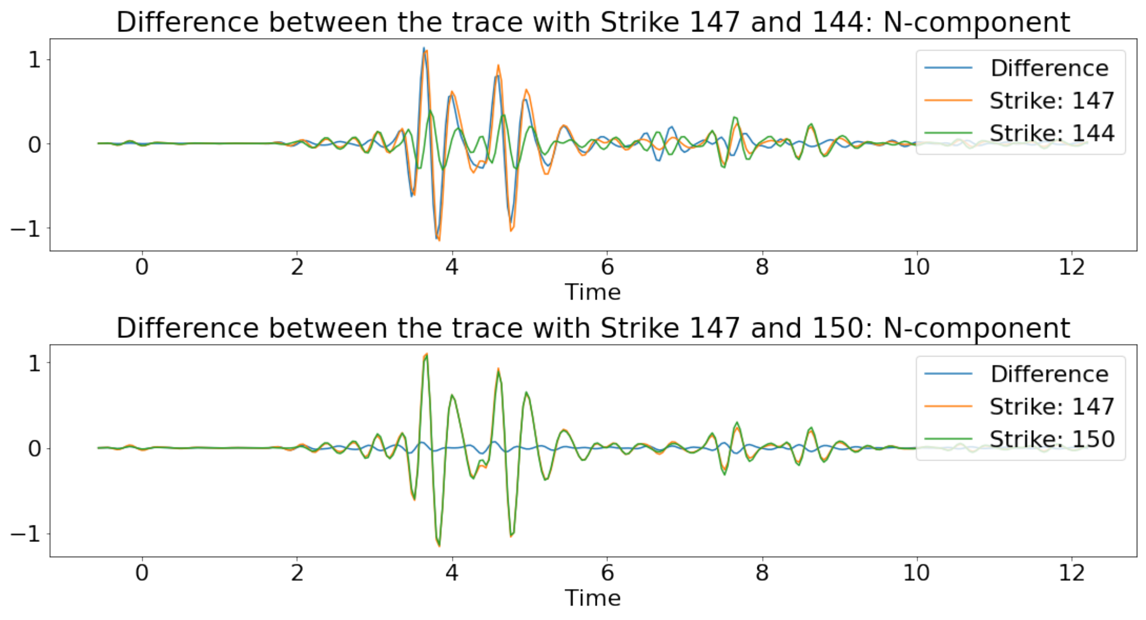


Figure A.3: Difference between two traces(green and orange), where the only difference is a small positive or negative change in the strike. The change is shown with by blue line.

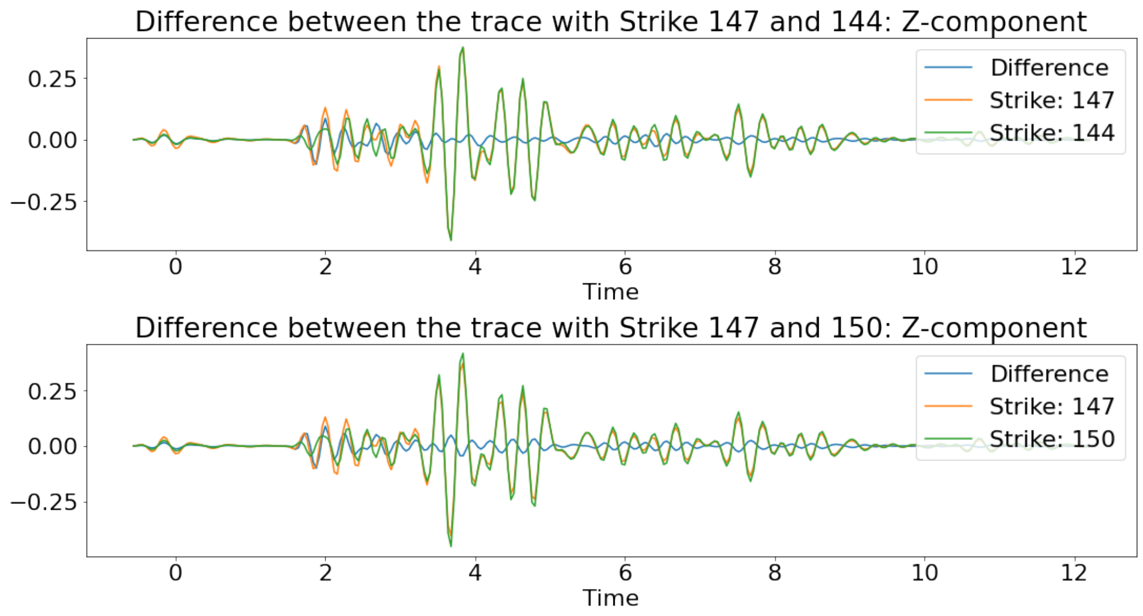


Figure A.4: Difference between two traces(green and orange), where the only difference is a small positive or negative change in the strike. The change is shown with by blue line.

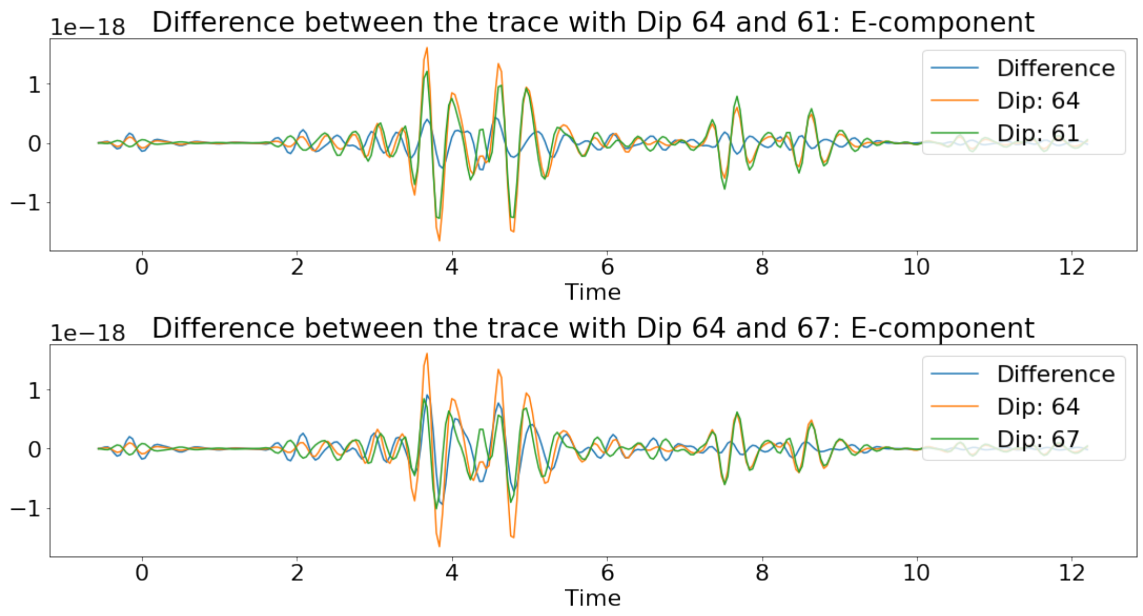


Figure A.5: Difference between two traces(green and orange), where the only difference is a small positive or negative change in the dip. The change is shown with by blue line.

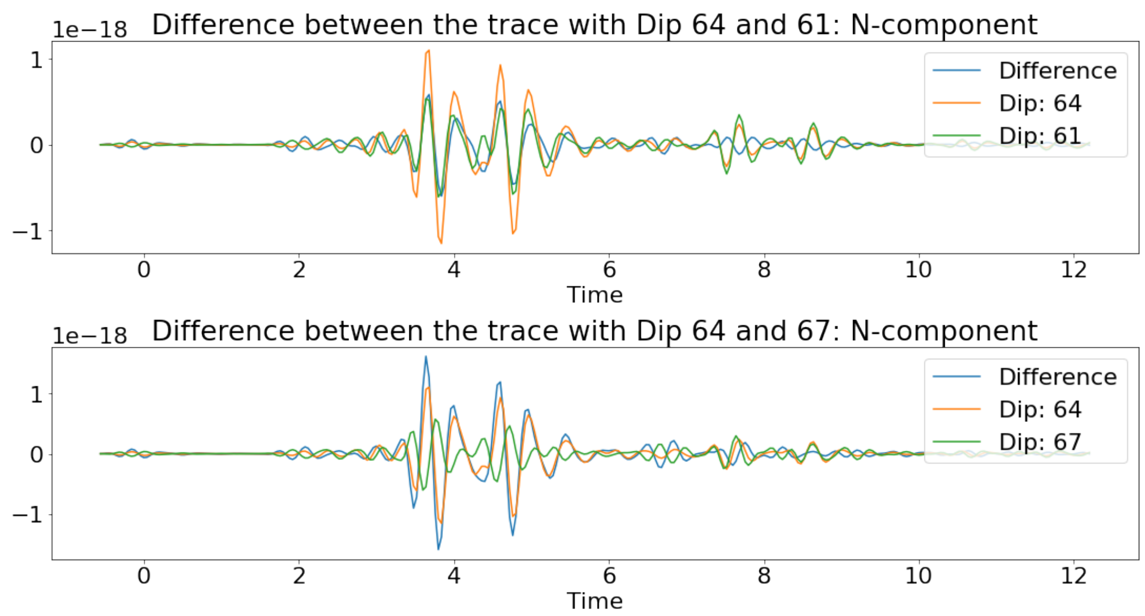


Figure A.6: Difference between two traces(green and orange), where the only difference is a small positive or negative change in the dip. The change is shown with by blue line.

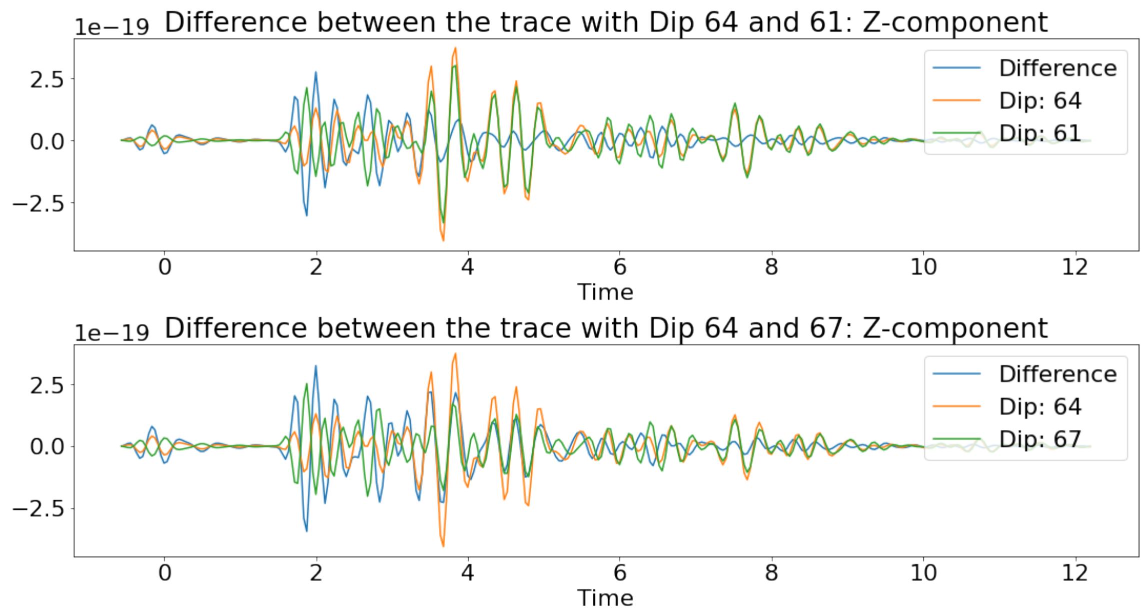


Figure A.7: Difference between two traces(green and orange), where the only difference is a small positive or negative change in the dip. The change is shown with by blue line.

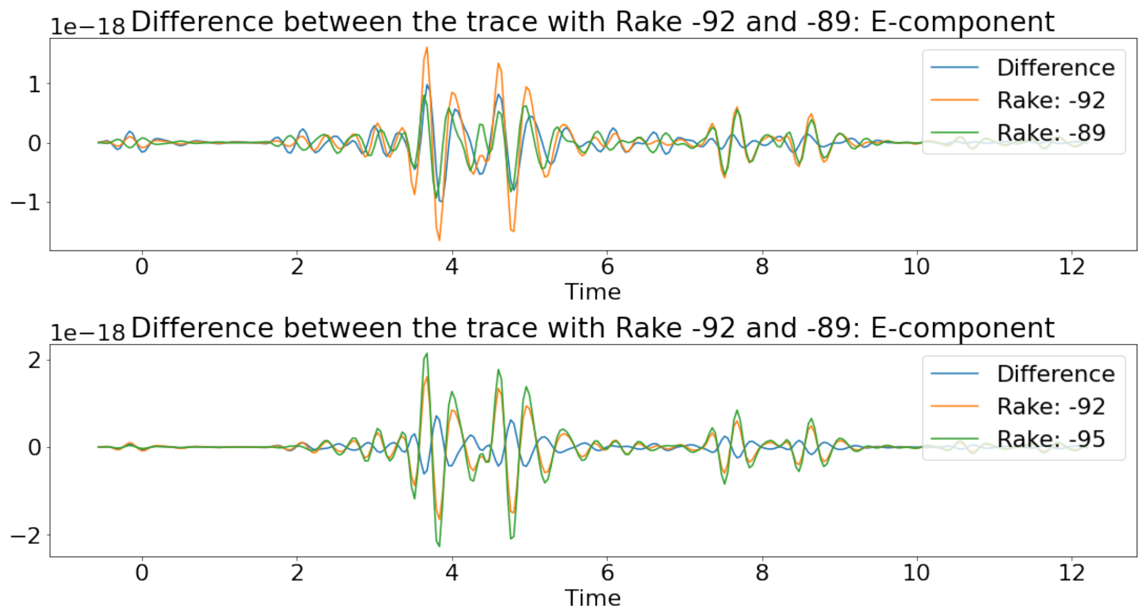


Figure A.8: Difference between two traces(green and orange), where the only difference is a small positive or negative change in the rake. The change is shown with by blue line.

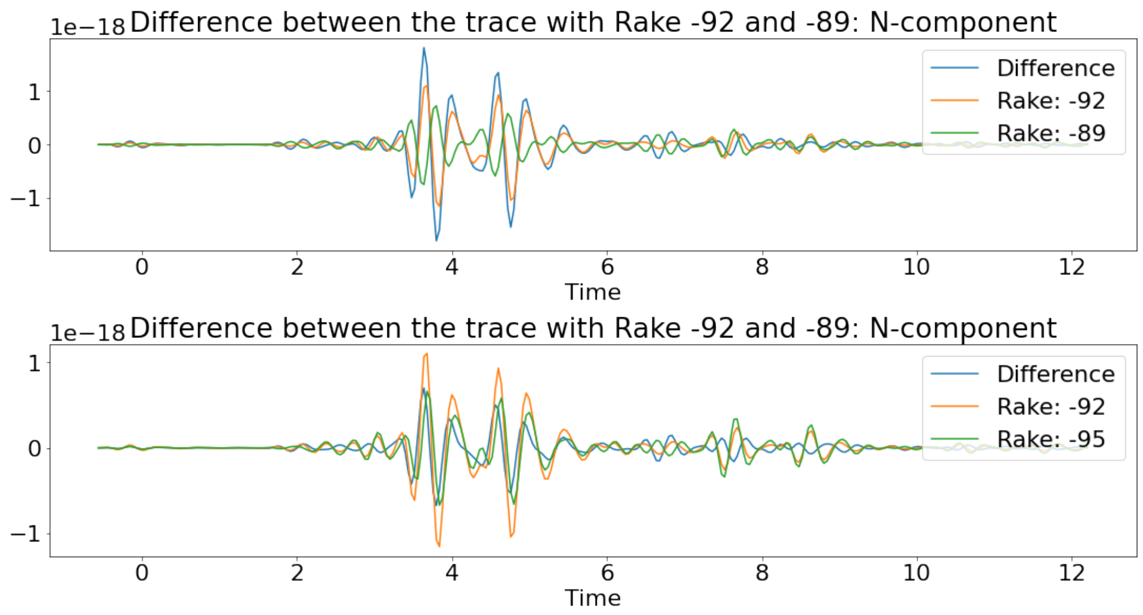


Figure A.9: Difference between two traces(green and orange), where the only difference is a small positive or negative change in the rake. The change is shown with by blue line.

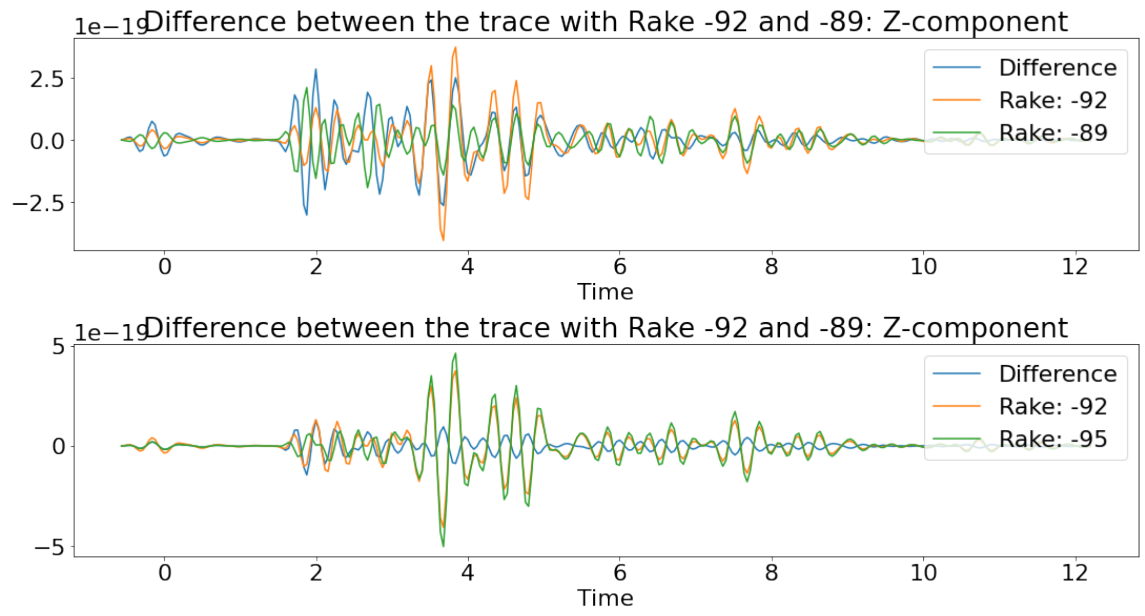


Figure A.10: Difference between two traces(green and orange), where the only difference is a small positive or negative change in the rake. The change is shown with by blue line.

A.3 Receiver Specific Changes

From their research, Kühn et al. (2020) concludes that the 150m depth receiver data works best for their inversion method. To see how much a signal changes due to these varying receiver depths, see Figures A.11 to A.13. Interestingly it is the N component that reacts most to the change in receiver depth. The Z-component also shows some changes, but it is not as strong as with the N-component data.

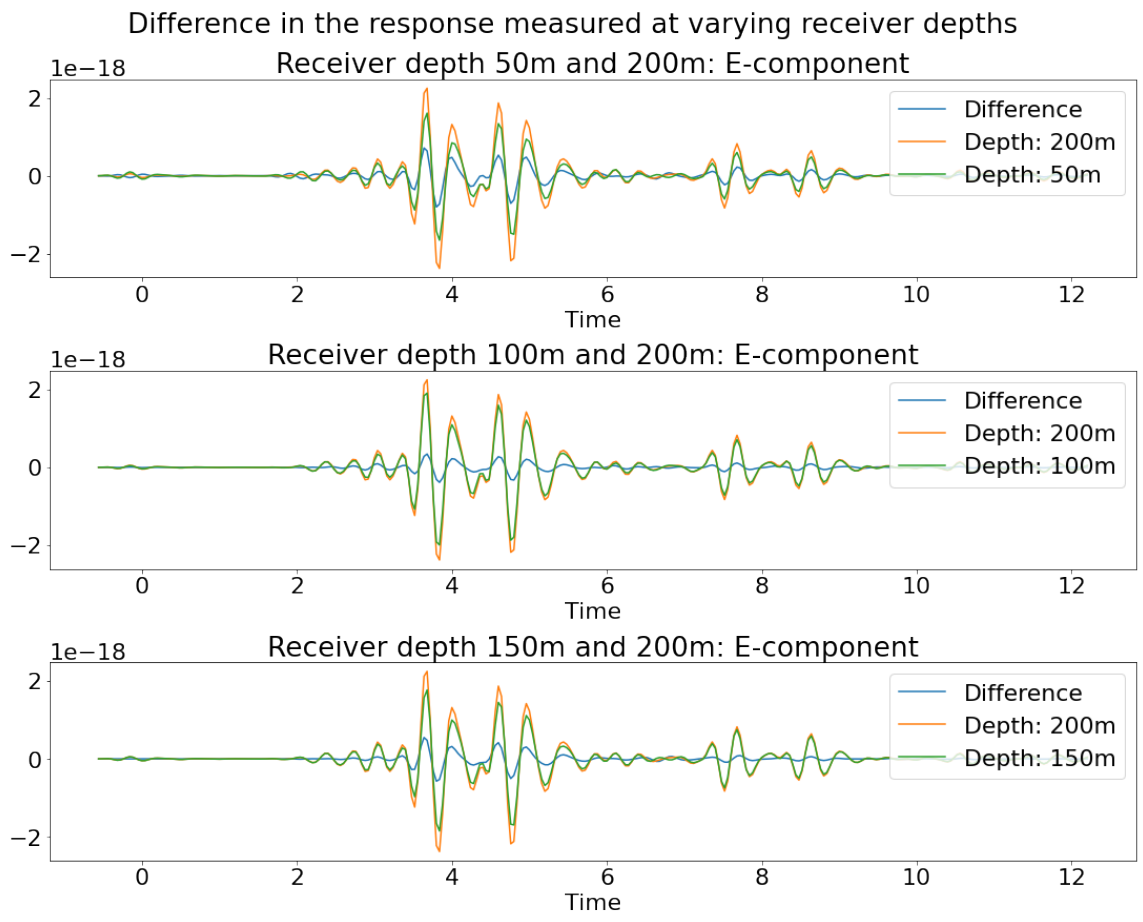


Figure A.11: Difference between two traces(green and orange), where the only difference is the depth of the receiver. The change is shown with by blue line.

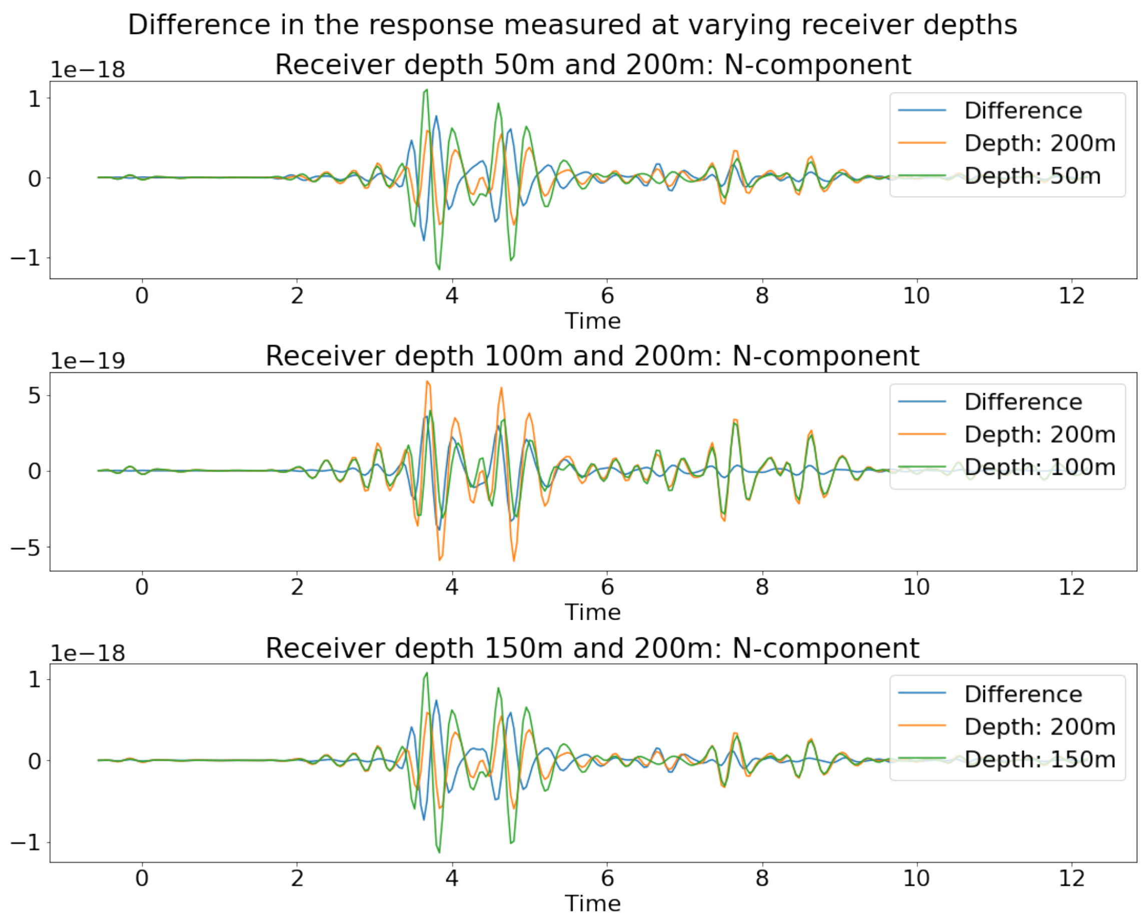


Figure A.12: Difference between two traces(green and orange), where the only difference is the depth of the receiver. The change is shown with by blue line.

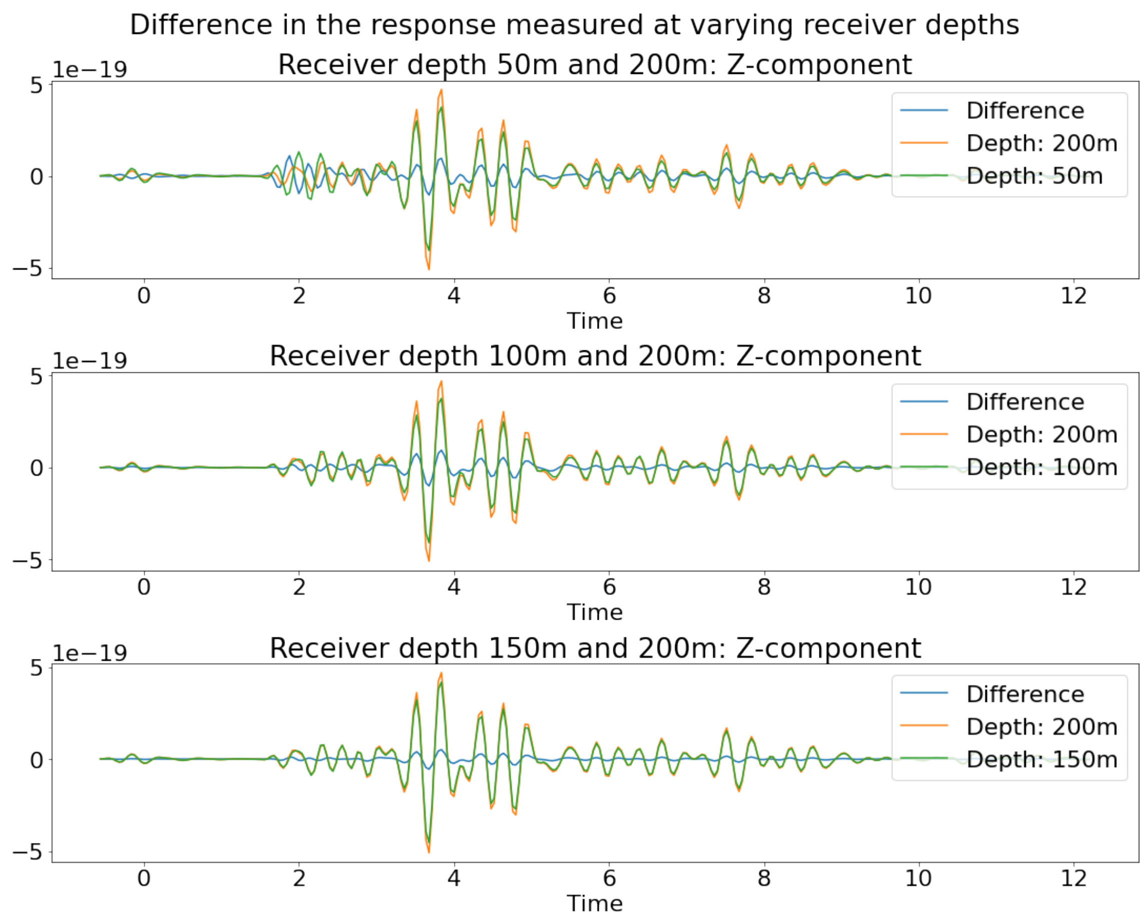


Figure A.13: Difference between two traces(green and orange), where the only difference is the depth of the receiver. The change is shown with by blue line.

Chapter B

Neural Network Configurations

This chapter shows all simulations done in order to investigate the trade-off between different hyperparameters in a FFNN (Table B.1) and MDN (Table B.2). The performance of the network is described by the overall EVS. This means that, instead of looking at the EVS score individually per MT component, the total EVS over all components has been calculated by the network. A consequence of this form of assessing the performance is that some combinations of hyperparameters in the network architecture might seem to not work (mostly for the heterogenous model). In reality this lower EVS score is caused by the inequality of sensitivity to the individual MT components.

B.1 FFNN

Table B.1: Network architectures tried for the hyperparameter finetuning of the problem presented in this thesis. This particular table shows the configuration set up and overall EVS scores for the FFNN. The overall EVS score includes the different performances of all independent MT components. This means that sometimes the overall EVS looks pretty bad, but in fact it is mainly the poor performing diagonal MT components that bring the overall score down.

Data Total	Learning Rate	Optimizer	Neurons First Hidden Layer	Neurons Second Hidden Layer	Mixtures	Regulizer	Regulizer Rate	Dropout	Dropout Rate	Activation Function	Batch Size	Input Scaled	Output Scaled	hom/ het	Overall EVS
10000	0,01	ADAM	50			no		yes	0,25	elu	750	1/max	[0,2pi]	hom	0,8212
10000	0,005	ADAM	50			no		yes	0,25	elu	750	1/max	[0,2pi]	hom	0,9120
10000	0,001	ADAM	50			no		yes	0,25	elu	750	1/max	[0,2pi]	hom	0,8977
10000	0,0005	ADAM	50			no		yes	0,25	elu	750	1/max	[0,2pi]	hom	0,8999
10000	0,0001	ADAM	50			no		yes	0,25	elu	750	1/max	[0,2pi]	hom	0,8966
10000	0,001	ADAM	50			no		yes	0,25	elu	250	1/max	[0,2pi]	hom	0,9123
10000	0,001	ADAM	50			no		yes	0,25	elu	500	1/max	[0,2pi]	hom	0,8980
10000	0,001	ADAM	50			no		yes	0,25	elu	1000	1/max	[0,2pi]	hom	0,8969
10000	0,001	ADAM	50			no		yes	0,25	elu	1250	1/max	[0,2pi]	hom	0,8858
10000	0,001	ADAM	50			no		yes	0,25	elu	1500	1/max	[0,2pi]	hom	0,8914
10000	0,001	ADAM	6			no		yes	0,25	elu	750	1/max	[0,2pi]	hom	0,5754
10000	0,001	ADAM	12			no		yes	0,25	elu	750	1/max	[0,2pi]	hom	0,8203
10000	0,001	ADAM	18			no		yes	0,25	elu	750	1/max	[0,2pi]	hom	0,8458

Table B.1 continued from previous page

Data Total	Learning Rate	Optimizer	Neurons First Hidden Layer	Neurons Second Hidden Layer	Mixtures	Regulizer	Regulizer Rate	Dropout	Dropout Rate	Activation Function	Batch Size	Input Scaled	Output Scaled	hom/ het	Overall EVS
10000	0,001	ADAM	30			no		yes	0,25	elu	750	1/max	[0,2pi]	hom	0,8924
10000	0,001	ADAM	100			no		yes	0,25	elu	750	1/max	[0,2pi]	hom	0,9050
10000	0,001	ADAM	150			no		yes	0,25	elu	750	1/max	[0,2pi]	hom	0,9171
10000	0,001	ADAM	200			no		yes	0,25	elu	750	1/max	[0,2pi]	hom	0,9118
10000	0,001	ADAM	250			no		yes	0,25	elu	750	1/max	[0,2pi]	hom	0,9341
10000	0,001	ADAM	300			no		yes	0,25	elu	750	1/max	[0,2pi]	hom	0,9518
10000	0,001	ADAM	350			no		yes	0,25	elu	750	1/max	[0,2pi]	hom	0,9516
10000	0,001	ADAM	50			no		yes	0,25	elu	750	1/max	[0,2pi]	hom	0,8994
10000	0,001	ADAM	50			no		yes	0,25	elu	750	1/max	[0,2pi]	hom	0,8930
10000	0,001	ADAM	50			no		yes	0,25	elu	750	1/max	[0,2pi]	hom	-11,0000
10000	0,001	ADAM	50			no		yes	0,25	elu	750	1/max	[0,2pi]	hom	0,8896
10000	0,001	ADAM	50			no		yes	0,25	elu	750	1/max	[0,2pi]	hom	-45,0000
10000	0,001	ADAM	50			no		yes	0,01	elu	750	1/max	[0,2pi]	hom	0,9491
10000	0,001	ADAM	50			no		yes	0,05	elu	750	1/max	[0,2pi]	hom	0,9387
10000	0,001	ADAM	50			no		yes	0,1	elu	750	1/max	[0,2pi]	hom	0,9236
10000	0,001	ADAM	50			no		yes	0,15	elu	750	1/max	[0,2pi]	hom	0,9042
10000	0,001	ADAM	50			no		yes	0,2	elu	750	1/max	[0,2pi]	hom	0,9004
10000	0,001	ADAM	50			no		yes	0,2	relu	750	1/max	[0,2pi]	hom	0,9076
10000	0,001	ADAM	50			no		yes	0,2	sofmtax	750	1/max	[0,2pi]	hom	0,1717
10000	0,001	ADAM	50			no		yes	0,2	sigmoid	750	1/max	[0,2pi]	hom	0,9447
10000	0,001	ADAM	50			no		yes	0,2	tanh	750	1/max	[0,2pi]	hom	0,9443
10000	0,001	ADAM	50			no		yes	0,2	selu	750	1/max	[0,2pi]	hom	0,9001
10000	0,001	ADAM	50			L1	0,0100	no		elu	750	1/max	[0,2pi]	hom	0,9143
10000	0,001	ADAM	50			L1	0,0500	no		elu	750	1/max	[0,2pi]	hom	0,9069
10000	0,001	ADAM	50			L1	0,0010	no		elu	750	1/max	[0,2pi]	hom	0,9202
10000	0,001	ADAM	50			L2	0,0100	no		elu	750	1/max	[0,2pi]	hom	0,9432
10000	0,001	ADAM	50			L2	0,0500	no		elu	750	1/max	[0,2pi]	hom	0,9149
10000	0,001	ADAM	50			L2	0,0010	no		elu	750	1/max	[0,2pi]	hom	0,9470
10000	0,001	ADAM	50			L1_L2	0,0100 0,0100	no		elu	750	1/max	[0,2pi]	hom	0,9154
10000	0,001	ADAM	50			L1_L2	0,0100 0,0500	no		elu	750	1/max	[0,2pi]	hom	0,9043

Table B.1 continued from previous page

Data Total	Learning Rate	Optimizer	Neurons First Hidden Layer	Neurons Second Hidden Layer	Mixtures	Regulizer	Regulizer Rate	Dropout	Dropout Rate	Activation Function	Batch Size	Input Scaled	Output Scaled	hom/ het	Overall EVS
10000	0,001	ADAM	50			L1_L2	0,0100 0,0050	no		elu	750	1/max	[0,2pi]	hom	0,9133
10000	0,001	ADAM	50			L1_L2	0,0500 0,0100	no		elu	750	1/max	[0,2pi]	hom	0,9076
10000	0,001	ADAM	50			L1_L2	0,0050 0,0100	no		elu	750	1/max	[0,2pi]	hom	0,9122
10000	0,001	ADAM	50			L1_L2	0,0010 0,0010	no		elu	750	1/max	[0,2pi]	hom	0,9417
10000	0,001	ADAM	50	20		no		yes	0.05	elu	750	1/max	[0,2pi]	hom	0,9488
10000	0,001	ADAM	50	50		no		yes	0.05	elu	750	1/max	[0,2pi]	hom	0,9599
10000	0,001	ADAM	50	100		no		yes	0.05	elu	750	1/max	[0,2pi]	hom	0,9607
10000	0,001	ADAM	50	150		no		yes	0.05	elu	750	1/max	[0,2pi]	hom	0,9628
10000	0,001	ADAM	50	200		no		yes	0.05	elu	750	1/max	[0,2pi]	hom	0,9643
10000	0,001	ADAM	100	20		no		yes	0.05	elu	750	1/max	[0,2pi]	hom	0,9471
10000	0,001	ADAM	100	50		no		yes	0.05	elu	750	1/max	[0,2pi]	hom	0,9601
10000	0,001	ADAM	100	100		no		yes	0.05	elu	750	1/max	[0,2pi]	hom	0,9649
10000	0,001	ADAM	100	150		no		yes	0.05	elu	750	1/max	[0,2pi]	hom	0,9650
10000	0,001	ADAM	100	200		no		yes	0.05	elu	750	1/max	[0,2pi]	hom	0,9643
10000	0,01	SGD	50			no		yes	0.05	elu	750	1/max	[0,2pi]	hom	0,9157
10000	0,005	SGD	50			no		yes	0.05	elu	750	1/max	[0,2pi]	hom	0,9140
10000	0,001	SGD	50			no		yes	0.05	elu	750	1/max	[0,2pi]	hom	0,9100
10000	0,0005	SGD	50			no		yes	0.05	elu	750	1/max	[0,2pi]	hom	0,9079
10000	0,0001	SGD	50			no		yes	0.05	elu	750	1/max	[0,2pi]	hom	0,8719
10000	0,01	SGD (0.2)	50			no		yes	0.05	elu	750	1/max	[0,2pi]	hom	0,9103
10000	0,01	SGD (0.4)	50			no		yes	0.05	elu	750	1/max	[0,2pi]	hom	0,9113
10000	0,01	SGD (0.6)	50			no		yes	0.05	elu	750	1/max	[0,2pi]	hom	0,9104
10000	0,01	SGD (0.8)	50			no		yes	0.05	elu	750	1/max	[0,2pi]	hom	0,9170
10000	0,01	SGD (0.99)	50			no		yes	0.05	elu	750	1/max	[0,2pi]	hom	0,9500

Table B.1 continued from previous page

Data Total	Learning Rate	Optimizer	Neurons First Hidden Layer	Neurons Second Hidden Layer	Mixtures	Regulizer	Regulizer Rate	Dropout	Dropout Rate	Activation Function	Batch Size	Input Scaled	Output Scaled	hom/ het	Overall EVS
10000	0,01	ADAM	50			no		yes	0,25	elu	750	yes	[0,2pi]	inhom	-4,4066
10000	0,005	ADAM	50			no		yes	0,25	elu	750	1/max	[0,2pi]	inhom	0,3724
10000	0,001	ADAM	50			no		yes	0,25	elu	750	1/max	[0,2pi]	inhom	0,5501
10000	0,0005	ADAM	50			no		yes	0,25	elu	750	1/max	[0,2pi]	inhom	0,6635
10000	0,0001	ADAM	50			no		yes	0,25	elu	750	1/max	[0,2pi]	inhom	0,7586
10000	0,000050	ADAM	50			no		yes	0,25	elu	750	1/max	[0,2pi]	inhom	0,7415
10000	0,000010	ADAM	50			no		yes	0,25	elu	750	1/max	[0,2pi]	inhom	0,6996
10000	0,000005	ADAM	50			no		yes	0,25	elu	750	1/max	[0,2pi]	inhom	0,7074
10000	0,000001	ADAM	50			no		yes	0,25	elu	750	1/max	[0,2pi]	inhom	0,5395
10000	0,001	ADAM	50			no		yes	0,25	elu	250	1/max	[0,2pi]	inhom	0,7167
10000	0,001	ADAM	50			no		yes	0,25	elu	500	1/max	[0,2pi]	inhom	0,7138
10000	0,001	ADAM	50			no		yes	0,25	elu	1000	1/max	[0,2pi]	inhom	0,5694
10000	0,001	ADAM	50			no		yes	0,25	elu	1250	1/max	[0,2pi]	inhom	0,2680
10000	0,001	ADAM	50			no		yes	0,25	elu	1500	1/max	[0,2pi]	inhom	0,7150
10000	0,001	ADAM	6			no		yes	0,25	elu	750	1/max	[0,2pi]	inhom	0,4411
10000	0,001	ADAM	12			no		yes	0,25	elu	750	1/max	[0,2pi]	inhom	0,3378
10000	0,001	ADAM	18			no		yes	0,25	elu	750	1/max	[0,2pi]	inhom	0,6924
10000	0,001	ADAM	30			no		yes	0,25	elu	750	1/max	[0,2pi]	inhom	0,6380
10000	0,001	ADAM	100			no		yes	0,25	elu	750	1/max	[0,2pi]	inhom	0,7182
10000	0,001	ADAM	150			no		yes	0,25	elu	750	1/max	[0,2pi]	inhom	0,6852
10000	0,001	ADAM	200			no		yes	0,25	elu	750	1/max	[0,2pi]	inhom	0,6844
10000	0,001	ADAM	250			no		yes	0,25	elu	750	1/max	[0,2pi]	inhom	0,6776
10000	0,001	ADAM	300			no		yes	0,25	elu	750	1/max	[0,2pi]	inhom	0,6717
10000	0,001	ADAM	350			no		yes	0,25	elu	750	1/max	[0,2pi]	inhom	0,6693
10000	0,001	ADAM	50			no		yes	0,25	elu	750	1/max	[0,2pi]	inhom	0,6622
10000	0,001	ADAM	50			no		yes	0,25	elu	750	1/max	[0,2pi]	inhom	0,7144
10000	0,001	ADAM	50			no		yes	0,25	elu	750	1/max	[0,2pi]	inhom	-0,9046
10000	0,001	ADAM	50			no		yes	0,25	elu	750	1/max	[0,2pi]	inhom	0,6873
10000	0,001	ADAM	50			no		yes	0,25	elu	750	1/max	[0,2pi]	inhom	-5,1733
10000	0,001	ADAM	50			no		yes	0,01	elu	750	1/max	[0,2pi]	inhom	0,6006
10000	0,001	ADAM	50			no		yes	0,05	elu	750	1/max	[0,2pi]	inhom	0,6330
10000	0,001	ADAM	50			no		yes	0,1	elu	750	1/max	[0,2pi]	inhom	0,6594

Table B.1 continued from previous page

Data Total	Learning Rate	Optimizer	Neurons First Hidden Layer	Neurons Second Hidden Layer	Mixtures	Regulizer	Regulizer Rate	Dropout	Dropout Rate	Activation Function	Batch Size	Input Scaled	Output Scaled	hom/ het	Overall EVS
10000	0,001	ADAM	50			no		yes	0,15	elu	750	1/max	[0,2pi]	inhom	0,7026
10000	0,001	ADAM	50			no		yes	0,2	elu	750	1/max	[0,2pi]	inhom	0,4885
10000	0,001	ADAM	50			no		yes	0,2	relu	750	1/max	[0,2pi]	inhom	0,6604
10000	0,001	ADAM	50			no		yes	0,2	sofmtax	750	1/max	[0,2pi]	inhom	0,0196
10000	0,001	ADAM	50			no		yes	0,2	sigmoid	750	1/max	[0,2pi]	inhom	0,8083
10000	0,001	ADAM	50			no		yes	0,2	tanh	750	1/max	[0,2pi]	inhom	0,7545
10000	0,001	ADAM	50			no		yes	0,2	selu	750	1/max	[0,2pi]	inhom	0,7436
10000	0,001	ADAM	50			L1	0,0100	no		elu	750	1/max	[0,2pi]	inhom	0,7320
10000	0,001	ADAM	50			L1	0,0500	no		elu	750	1/max	[0,2pi]	inhom	0,6281
10000	0,001	ADAM	50			L1	0,0010	no		elu	750	1/max	[0,2pi]	inhom	0,7327
10000	0,001	ADAM	50			L2	0,0100	no		elu	750	1/max	[0,2pi]	inhom	0,7054
10000	0,001	ADAM	50			L2	0,0500	no		elu	750	1/max	[0,2pi]	inhom	0,7520
10000	0,001	ADAM	50			L2	0,0010	no		elu	750	1/max	[0,2pi]	inhom	0,6645
10000	0,001	ADAM	50			L1_L2	0,0100 0,0100	no		elu	750	1/max	[0,2pi]	inhom	0,7365
10000	0,001	ADAM	50			L1_L2	0,0100 0,0500	no		elu	750	1/max	[0,2pi]	inhom	0,7187
10000	0,001	ADAM	50			L1_L2	0,0100 0,0050	no		elu	750	1/max	[0,2pi]	inhom	0,7271
10000	0,001	ADAM	50			L1_L2	0,0500 0,0100	no		elu	750	1/max	[0,2pi]	inhom	0,6876
10000	0,001	ADAM	50			L1_L2	0,0050 0,0100	no		elu	750	1/max	[0,2pi]	inhom	0,7345
10000	0,001	ADAM	50			L1_L2	0,0010 0,0010	no		elu	750	1/max	[0,2pi]	inhom	0,7241
10000	0,001	ADAM	50	20		no		yes	0.05	elu	750	1/max	[0,2pi]	inhom	0,7991
10000	0,001	ADAM	50	50		no		yes	0.05	elu	750	1/max	[0,2pi]	inhom	0,8105
10000	0,001	ADAM	50	100		no		yes	0.05	elu	750	1/max	[0,2pi]	inhom	0,8111
10000	0,001	ADAM	50	150		no		yes	0.05	elu	750	1/max	[0,2pi]	inhom	0,8244
10000	0,001	ADAM	50	200		no		yes	0.05	elu	750	1/max	[0,2pi]	inhom	0,8083
10000	0,001	ADAM	100	20		no		yes	0.05	elu	750	1/max	[0,2pi]	inhom	0,7671
10000	0,001	ADAM	100	50		no		yes	0.05	elu	750	1/max	[0,2pi]	inhom	0,7816

Table B.1 continued from previous page

Data Total	Learning Rate	Optimizer	Neurons First Hidden Layer	Neurons Second Hidden Layer	Mixtures	Regulizer	Regulizer Rate	Dropout	Dropout Rate	Activation Function	Batch Size	Input Scaled	Output Scaled	hom/ het	Overall EVS
10000	0,001	ADAM	100	100		no		yes	0.05	elu	750	1/max	[0,2pi]	inhom	0,7879
10000	0,001	ADAM	100	150		no		yes	0.05	elu	750	1/max	[0,2pi]	inhom	0,8105
10000	0,001	ADAM	100	200		no		yes	0.05	elu	750	1/max	[0,2pi]	inhom	0,8133
10000	0,01	SGD	50			no		yes	0.05	elu	750	1/max	[0,2pi]	inhom	0,6858
10000	0,005	SGD	50			no		yes	0.05	elu	750	1/max	[0,2pi]	inhom	0,6452
10000	0,001	SGD	50			no		yes	0.05	elu	750	1/max	[0,2pi]	inhom	0,4287
10000	0,0005	SGD	50			no		yes	0.05	elu	750	1/max	[0,2pi]	inhom	0,3313
10000	0,0001	SGD	50			no		yes	0.05	elu	750	1/max	[0,2pi]	inhom	0,1552
10000	0,01	SGD (0.2)	50			no		yes	0.05	elu	750	1/max	[0,2pi]	inhom	0,4658
10000	0,01	SGD (0.4)	50			no		yes	0.05	elu	750	1/max	[0,2pi]	inhom	0,5101
10000	0,01	SGD (0.6)	50			no		yes	0.05	elu	750	1/max	[0,2pi]	inhom	0,5637
10000	0,01	SGD (0.8)	50			no		yes	0.05	elu	750	1/max	[0,2pi]	inhom	0,6436
10000	0,01	SGD (0.99)	50			no		yes	0.05	elu	750	1/max	[0,2pi]	inhom	0,7369

B.2 MDN

Table B.2: Network architectures tried for the hyperparameter finetuning of the problem presented in this thesis. This particular table shows the configuration set up and overall EVS scores for the MDN. The overall EVS score includes the different performances of all independent MT components. This means that sometimes the overall EVS looks pretty bad, but in fact it is mainly the poor performing diagonal MT components that bring the overall score down.

Data Total	Learning Rate	Optimizer	Neurons First Hidden Layer	Neurons Second Hidden Layer	Mixtures	Regulizer	Regulizer Rate	Dropout	Dropout Rate	Activation Function	Batch Size	Input Scaled	Output Scaled	hom/ het	Overall EVS
10000	0,01	ADAM	50		1000	no		yes	0,25	elu	750	1/max	[0,2pi]	hom	-14.991
10000	0,005	ADAM	50		60	no		yes	0,25	elu	750	1/max	[0,2pi]	hom	0,6942
10000	0,001	ADAM	50		60	no		yes	0,25	elu	750	1/max	[0,2pi]	hom	0,7860
10000	0,0005	ADAM	50		60	no		yes	0,25	elu	750	1/max	[0,2pi]	hom	0,7837
10000	0,0001	ADAM	50		60	no		yes	0,25	elu	750	1/max	[0,2pi]	hom	0,7232
10000	0,001	ADAM	50		60	no		yes	0,25	elu	250	1/max	[0,2pi]	hom	0,7961
10000	0,001	ADAM	50		60	no		yes	0,25	elu	500	1/max	[0,2pi]	hom	0,7747
10000	0,001	ADAM	50		60	no		yes	0,25	elu	1000	1/max	[0,2pi]	hom	0,7703
10000	0,001	ADAM	50		60	no		yes	0,25	elu	1250	1/max	[0,2pi]	hom	0,7317
10000	0,001	ADAM	50		60	no		yes	0,25	elu	1500	1/max	[0,2pi]	hom	0,7764
10000	0,001	ADAM	6		60	no		yes	0,25	elu	750	1/max	[0,2pi]	hom	0,7636
10000	0,001	ADAM	12		60	no		yes	0,25	elu	750	1/max	[0,2pi]	hom	0,5953
10000	0,001	ADAM	18		60	no		yes	0,25	elu	750	1/max	[0,2pi]	hom	0,6579
10000	0,001	ADAM	30		60	no		yes	0,25	elu	750	1/max	[0,2pi]	hom	0,7216
10000	0,001	ADAM	100		60	no		yes	0,25	elu	750	1/max	[0,2pi]	hom	0,8458
10000	0,001	ADAM	150		60	no		yes	0,25	elu	750	1/max	[0,2pi]	hom	0,8557
10000	0,001	ADAM	200		60	no		yes	0,25	elu	750	1/max	[0,2pi]	hom	0,8773
10000	0,001	ADAM	250		60	no		yes	0,25	elu	750	1/max	[0,2pi]	hom	0,8792
10000	0,001	ADAM	300		60	no		yes	0,25	elu	750	1/max	[0,2pi]	hom	0,8866
10000	0,001	ADAM	350		60	no		yes	0,25	elu	750	1/max	[0,2pi]	hom	0,8811
10000	0,001	ADAM	50		6	no		yes	0,25	elu	750	1/max	[0,2pi]	hom	-605
10000	0,001	ADAM	50		12	no		yes	0,25	elu	750	1/max	[0,2pi]	hom	-604
10000	0,001	ADAM	50		18	no		yes	0,25	elu	750	1/max	[0,2pi]	hom	0,7912
10000	0,001	ADAM	50		30	no		yes	0,25	elu	750	1/max	[0,2pi]	hom	0,7900
10000	0,001	ADAM	50		100	no		yes	0,25	elu	750	1/max	[0,2pi]	hom	0,7536
10000	0,001	ADAM	50		60	no		yes	0,01	elu	750	1/max	[0,2pi]	hom	0,8602
10000	0,001	ADAM	50		60	no		yes	0,05	elu	750	1/max	[0,2pi]	hom	0,8802

Table B.2 continued from previous page

Data Total	Learning Rate	Optimizer	Neurons First Hidden Layer	Neurons Second Hidden Layer	Mixtures	Regulizer	Regulizer Rate	Dropout	Dropout Rate	Activation Function	Batch Size	Input Scaled	Output Scaled	hom/ het	Overall EVS
10000	0,001	ADAM	50		60	no		yes	0,1	elu	750	1/max	[0,2pi]	hom	0,8465
10000	0,001	ADAM	50		60	no		yes	0,15	elu	750	1/max	[0,2pi]	hom	0,7907
10000	0,001	ADAM	50		60	no		yes	0,2	elu	750	1/max	[0,2pi]	hom	0,7729
10000	0,001	ADAM	50		60	no		yes	0,2	relu	750	1/max	[0,2pi]	hom	0,7842
10000	0,001	ADAM	50		60	no		yes	0,2	sofmtax	750	1/max	[0,2pi]	hom	-0,6686
10000	0,001	ADAM	50		60	no		yes	0,2	sigmoid	750	1/max	[0,2pi]	hom	0,8216
10000	0,001	ADAM	50		60	no		yes	0,2	tanh	750	1/max	[0,2pi]	hom	0,7040
10000	0,001	ADAM	50		60	no		yes	0,2	selu	750	1/max	[0,2pi]	hom	0,8256
10000	0,001	ADAM	50		60	L1	0,0100	no		elu	750	1/max	[0,2pi]	hom	0,8812
10000	0,001	ADAM	50		60	L1	0,0500	no		elu	750	1/max	[0,2pi]	hom	0,8656
10000	0,001	ADAM	50		60	L1	0,0010	no		elu	750	1/max	[0,2pi]	hom	0,8856
10000	0,001	ADAM	50		60	L2	0,0100	no		elu	750	1/max	[0,2pi]	hom	0,8610
10000	0,001	ADAM	50		60	L2	0,0500	no		elu	750	1/max	[0,2pi]	hom	0,8450
10000	0,001	ADAM	50		60	L2	0,0010	no		elu	750	1/max	[0,2pi]	hom	0,8724
10000	0,001	ADAM	50		60	L1_L2	0,0100 0,0100	no		elu	750	1/max	[0,2pi]	hom	0,8804
10000	0,001	ADAM	50		60	L1_L2	0,0100 0,0500	no		elu	750	1/max	[0,2pi]	hom	0,8817
10000	0,001	ADAM	50		60	L1_L2	0,0100 0,0050	no		elu	750	1/max	[0,2pi]	hom	0,8880
10000	0,001	ADAM	50		60	L1_L2	0,0500 0.0100	no		elu	750	1/max	[0,2pi]	hom	0,8837
10000	0,001	ADAM	50		60	L1_L2	0,0050 0,0100	no		elu	750	1/max	[0,2pi]	hom	0,8878
10000	0,001	ADAM	50		60	L1_L2	0,0010 0,0010	no		elu	750	1/max	[0,2pi]	hom	0,8809
10000	0,001	ADAM	50	20	60	no		yes	0.05	elu	750	1/max	[0,2pi]	hom	0,8459
10000	0,001	ADAM	50	50	60	no		yes	0.05	elu	750	1/max	[0,2pi]	hom	0,8925
10000	0,001	ADAM	50	100	60	no		yes	0.05	elu	750	1/max	[0,2pi]	hom	0,9098
10000	0,001	ADAM	50	150	60	no		yes	0.05	elu	750	1/max	[0,2pi]	hom	0,9082
10000	0,001	ADAM	50	200	60	no		yes	0.05	elu	750	1/max	[0,2pi]	hom	0,9056
10000	0,001	ADAM	100	20	60	no		yes	0.05	elu	750	1/max	[0,2pi]	hom	0,8893

Table B.2 continued from previous page

Data Total	Learning Rate	Optimizer	Neurons First Hidden Layer	Neurons Second Hidden Layer	Mixtures	Regulizer	Regulizer Rate	Dropout	Dropout Rate	Activation Function	Batch Size	Input Scaled	Output Scaled	hom/ het	Overall EVS
10000	0,001	ADAM	100	50	60	no		yes	0.05	elu	750	1/max	[0,2pi]	hom	0,8975
10000	0,001	ADAM	100	100	60	no		yes	0.05	elu	750	1/max	[0,2pi]	hom	0,9208
10000	0,001	ADAM	100	150	60	no		yes	0.05	elu	750	1/max	[0,2pi]	hom	0,9143
10000	0,001	ADAM	100	200	60	no		yes	0.05	elu	750	1/max	[0,2pi]	hom	0,9200
10000	0,01	SGD	50		60	no		yes	0.05	elu	750	1/max	[0,2pi]	hom	0,8097
10000	0,005	SGD	50		60	no		yes	0.05	elu	750	1/max	[0,2pi]	hom	0,8090
10000	0,001	SGD	50		60	no		yes	0.05	elu	750	1/max	[0,2pi]	hom	0,7416
10000	0,0005	SGD	50		60	no		yes	0.05	elu	750	1/max	[0,2pi]	hom	0,7715
10000	0,0001	SGD	50		60	no		yes	0.05	elu	750	1/max	[0,2pi]	hom	-0,6325
10000	0,01	SGD (0.2)	50		60	no		yes	0.05	elu	750	1/max	[0,2pi]	hom	0,7997
10000	0,01	SGD (0.4)	50		60	no		yes	0.05	elu	750	1/max	[0,2pi]	hom	0,8126
10000	0,01	SGD (0.6)	50		60	no		yes	0.05	elu	750	1/max	[0,2pi]	hom	0,7986
10000	0,01	SGD (0.8)	50		60	no		yes	0.05	elu	750	1/max	[0,2pi]	hom	0,8491
10000	0,01	SGD (0.99)	50		60	no		yes	0.05	elu	750	1/max	[0,2pi]	hom	0,8333
10000	0,01	ADAM	50		1000	no		yes	0,25	elu	750	yes	[0,2pi]	inhom	-4.062
10000	0,005	ADAM	50		60	no		yes	0,25	elu	750	1/max	[0,2pi]	inhom	-0,2292
10000	0,001	ADAM	50		60	no		yes	0,25	elu	750	1/max	[0,2pi]	inhom	0,5639
10000	0,0005	ADAM	50		60	no		yes	0,25	elu	750	1/max	[0,2pi]	inhom	0,6040
10000	0,0001	ADAM	50		60	no		yes	0,25	elu	750	1/max	[0,2pi]	inhom	0,4708
10000	0,001	ADAM	50		60	no		yes	0,25	elu	250	1/max	[0,2pi]	inhom	0,4025
10000	0,001	ADAM	50		60	no		yes	0,25	elu	500	1/max	[0,2pi]	inhom	0,5079
10000	0,001	ADAM	50		60	no		yes	0,25	elu	1000	1/max	[0,2pi]	inhom	0,4441
10000	0,001	ADAM	50		60	no		yes	0,25	elu	1250	1/max	[0,2pi]	inhom	0,5129
10000	0,001	ADAM	50		60	no		yes	0,25	elu	1500	1/max	[0,2pi]	inhom	0,6212
10000	0,001	ADAM	6		60	no		yes	0,25	elu	750	1/max	[0,2pi]	inhom	-0,1107
10000	0,001	ADAM	12		60	no		yes	0,25	elu	750	1/max	[0,2pi]	inhom	0,0656
10000	0,001	ADAM	18		60	no		yes	0,25	elu	750	1/max	[0,2pi]	inhom	0,2810

Table B.2 continued from previous page

Data Total	Learning Rate	Optimizer	Neurons First Hidden Layer	Neurons Second Hidden Layer	Mixtures	Regulizer	Regulizer Rate	Dropout	Dropout Rate	Activation Function	Batch Size	Input Scaled	Output Scaled	hom/ het	Overall EVS
10000	0,001	ADAM	30		60	no		yes	0,25	elu	750	1/max	[0,2pi]	inhom	0,3887
10000	0,001	ADAM	100		60	no		yes	0,25	elu	750	1/max	[0,2pi]	inhom	0,5297
10000	0,001	ADAM	150		60	no		yes	0,25	elu	750	1/max	[0,2pi]	inhom	0,5491
10000	0,001	ADAM	200		60	no		yes	0,25	elu	750	1/max	[0,2pi]	inhom	0,5059
10000	0,001	ADAM	250		60	no		yes	0,25	elu	750	1/max	[0,2pi]	inhom	0,4554
10000	0,001	ADAM	300		60	no		yes	0,25	elu	750	1/max	[0,2pi]	inhom	0,4377
10000	0,001	ADAM	350		60	no		yes	0,25	elu	750	1/max	[0,2pi]	inhom	0,5147
10000	0,001	ADAM	50		6	no		yes	0,25	elu	750	1/max	[0,2pi]	inhom	-84,0000
10000	0,001	ADAM	50		12	no		yes	0,25	elu	750	1/max	[0,2pi]	inhom	-67,0000
10000	0,001	ADAM	50		18	no		yes	0,25	elu	750	1/max	[0,2pi]	inhom	0,4998
10000	0,001	ADAM	50		30	no		yes	0,25	elu	750	1/max	[0,2pi]	inhom	0,4667
10000	0,001	ADAM	50		100	no		yes	0,25	elu	750	1/max	[0,2pi]	inhom	0,4511
10000	0,001	ADAM	50		60	no		yes	0,01	elu	750	1/max	[0,2pi]	inhom	0,2524
10000	0,001	ADAM	50		60	no		yes	0,05	elu	750	1/max	[0,2pi]	inhom	0,4132
10000	0,001	ADAM	50		60	no		yes	0,1	elu	750	1/max	[0,2pi]	inhom	0,5362
10000	0,001	ADAM	50		60	no		yes	0,15	elu	750	1/max	[0,2pi]	inhom	0,4654
10000	0,001	ADAM	50		60	no		yes	0,2	elu	750	1/max	[0,2pi]	inhom	0,5701
10000	0,001	ADAM	50		60	no		yes	0,2	relu	750	1/max	[0,2pi]	inhom	0,4605
10000	0,001	ADAM	50		60	no		yes	0,2	sofmtax	750	1/max	[0,2pi]	inhom	-0,9187
10000	0,001	ADAM	50		60	no		yes	0,2	sigmoid	750	1/max	[0,2pi]	inhom	0,5351
10000	0,001	ADAM	50		60	no		yes	0,2	tanh	750	1/max	[0,2pi]	inhom	0,0874
10000	0,001	ADAM	50		60	no		yes	0,2	selu	750	1/max	[0,2pi]	inhom	0,5268
10000	0,001	ADAM	50		60	L1	0,0100	no		elu	750	1/max	[0,2pi]	inhom	0,1829
10000	0,001	ADAM	50		60	L1	0,0500	no		elu	750	1/max	[0,2pi]	inhom	0,0560
10000	0,001	ADAM	50		60	L1	0,0010	no		elu	750	1/max	[0,2pi]	inhom	0,0818
10000	0,001	ADAM	50		60	L2	0,0100	no		elu	750	1/max	[0,2pi]	inhom	0,2682
10000	0,001	ADAM	50		60	L2	0,0500	no		elu	750	1/max	[0,2pi]	inhom	0,3194
10000	0,001	ADAM	50		60	L2	0,0010	no		elu	750	1/max	[0,2pi]	inhom	0,2631
10000	0,001	ADAM	50		60	L1_L2	0,0100 0,0100	no		elu	750	1/max	[0,2pi]	inhom	0,0489
10000	0,001	ADAM	50		60	L1_L2	0,0100 0,0500	no		elu	750	1/max	[0,2pi]	inhom	0,1457

Table B.2 continued from previous page

Data Total	Learning Rate	Optimizer	Neurons First Hidden Layer	Neurons Second Hidden Layer	Mixtures	Regulizer	Regulizer Rate	Dropout	Dropout Rate	Activation Function	Batch Size	Input Scaled	Output Scaled	hom/ het	Overall EVS
10000	0,001	ADAM	50		60	L1_L2	0,0100 0,0050	no		elu	750	1/max	[0,2pi]	inhom	0,0714
10000	0,001	ADAM	50		60	L1_L2	0,0500 0,0100	no		elu	750	1/max	[0,2pi]	inhom	0,0937
10000	0,001	ADAM	50		60	L1_L2	0,0050 0,0100	no		elu	750	1/max	[0,2pi]	inhom	0,0109
10000	0,001	ADAM	50		60	L1_L2	0,0010 0,0010	no		elu	750	1/max	[0,2pi]	inhom	0,2202
100000	0,001	ADAM	100		60	no		yes	0.05	elu	750	1/max	[0,2pi]	inhom	-0,7253
10000	0,001	ADAM	50	20	60	no		yes	0.05	elu	750	1/max	[0,2pi]	inhom	0,5521
10000	0,001	ADAM	50	50	60	no		yes	0.05	elu	750	1/max	[0,2pi]	inhom	0,6165
10000	0,001	ADAM	50	100	60	no		yes	0.05	elu	750	1/max	[0,2pi]	inhom	0,6208
10000	0,001	ADAM	50	150	60	no		yes	0.05	elu	750	1/max	[0,2pi]	inhom	0,6273
10000	0,001	ADAM	50	200	60	no		yes	0.05	elu	750	1/max	[0,2pi]	inhom	0,6500
10000	0,001	ADAM	100	20	60	no		yes	0.05	elu	750	1/max	[0,2pi]	inhom	0,4057
10000	0,001	ADAM	100	50	60	no		yes	0.05	elu	750	1/max	[0,2pi]	inhom	0,4992
10000	0,001	ADAM	100	100	60	no		yes	0.05	elu	750	1/max	[0,2pi]	inhom	0,5395
10000	0,001	ADAM	100	150	60	no		yes	0.05	elu	750	1/max	[0,2pi]	inhom	0,5570
10000	0,001	ADAM	100	200	60	no		yes	0.05	elu	750	1/max	[0,2pi]	inhom	0,5921
10000	0,01	SGD	50		60	no		yes	0.05	elu	750	1/max	[0,2pi]	inhom	0,1935
10000	0,005	SGD	50		60	no		yes	0.05	elu	750	1/max	[0,2pi]	inhom	0,1301
10000	0,001	SGD	50		60	no		yes	0.05	elu	750	1/max	[0,2pi]	inhom	-0,1461
10000	0,0005	SGD	50		60	no		yes	0.05	elu	750	1/max	[0,2pi]	inhom	-0,5136
10000	0,0001	SGD	50		60	no		yes	0.05	elu	750	1/max	[0,2pi]	inhom	-0,9598
10000	0,01	SGD (0.2)	50		60	no		yes	0.05	elu	750	1/max	[0,2pi]	inhom	-0,0277
10000	0,01	SGD (0.4)	50		60	no		yes	0.05	elu	750	1/max	[0,2pi]	inhom	0,1038
10000	0,01	SGD (0.6)	50		60	no		yes	0.05	elu	750	1/max	[0,2pi]	inhom	0,1713
10000	0,01	SGD (0.8)	50		60	no		yes	0.05	elu	750	1/max	[0,2pi]	inhom	0,2027

Table B.2 continued from previous page

Data Total	Learning Rate	Optimizer	Neurons First Hidden Layer	Neurons Second Hidden Layer	Mixtures	Regulizer	Regulizer Rate	Dropout	Dropout Rate	Activation Function	Batch Size	Input Scaled	Output Scaled	hom/ het	Overall EVS
10000	0,01	SGD (0.99)	50		60	no		yes	0.05	elu	750	1/max	[0,2pi]	inhom	0,3772