Localisation of scattering objects using neural networks

MSc Thesis
May 2021

Domonkos Haffner
Physics MSc Student

Supervisor:

Ferenc Izsák, associate professor
Mathematical Institute, Department of Applied Analysis and Computational Mathematics

Eötvös Loránd University, Faculty of Natural Sciences

Contents

Chapter 1

Introduction

In recent years, the use of neural networks has become widespread. The main direction is the recognition, processing and analysis of images and texts with efficient optimisation algorithms. The computer background capable of processing many data also contributes to the fact that the method has become one of the most important research areas and a modern method in close connection with data science. Such procedures also provide the basis for modern autonomous learning systems. There are relatively few theoretical analytical works in this field, which could identify the limitations and major difficulties of the applications. The compilation of the web itself is also intuitive: it is based on experience and some general observations. Accordingly, the literature is not well-structured either, and interesting, importantly detailed writings often appear as conference papers and internal reports, and they mostly cover a variety of case studies that provide well-functioning methods. This is obviously the goal itself, so the question is missing, why these work so well and why each particular neural network should be used. Among these tasks, there are very few predictions and simulations of physical systems, which are usually described by a continuous mathematical model. That is the reason we undertook to solve a task like this.

We have chosen a physical problem which is modelled by a partial differential equation. Usually partial differential equations can not be solved easily, however, we know the solution for the chosen one; we are looking for the parameters in the equations or the range in which the equation is given. Mathematical theory does not guarantee the clarity of the searched domain forms, nor its continuous dependence on the data. These two facts cause great difficulties in practice. If there is no clear domain form, then an approximation can be used as an optimal value, which can make any optimisation procedure uncertain. On the other hand, if the data is artificially given some noise, the range of results can be very different.

In this case, solving our problem means a numerical approximation for the solution. For this, we used the Matlab software package and then the Python program, including the Keras library system developed for neural networks. The relatively easy usability of the Python programming language and the extensive system of subroutines written there in Keras are required to generate fast and flexible code with a transparent structure. Of course, all this is possible if a suitable data set is available for training the neural network (i.e., practically optimising the parameters within it). We use Matlab simulations to produce the required data set.

The research was a success; the desired Neural Network was created to solve the localisation problem. The results are published in MDPI [Q1] Sensors. [7]

Chapter 2

Description of the model

In the first section of this chapter, the physical problem is described. Assumptions are listed, which will be used in the physical and mathematical modelling of the given problem. The corresponding equations and the corresponding notations are given in the second section of this chapter.

2.1 Description of the physical problem

We investigate a kind of inverse scattering problem. In general, these problems involve the determination of the physical properties of a scattering medium. In most cases, the location and material properties of some scattering objects - which can highly influence the scattering - are of interest. As an input data, we usually have an known incident wave and a real measured data consisting the sum of the incident and the scattered wave.

Turning to our specific case, we assume a homogeneous medium in a 2-dimensional space segment with known transmission coefficient.

We assume here the presence of two different but identical circles. A particular incident wave is applied to medium, which can be a single point wave or even a plane wave. This wave is then scattered on, or reflected from the spheres. Often, scattering is more dominant, hence the spheres are called scattering centres.

This scattering phenomenon is significantly influenced by the structure of the spheres: whether they absorb the wave, generate vibrations in them, or just completely reflect them.

The total wave containing the incident and the scattered wave in a simulation is shown in figure 4.2.2 and 4.2.3.

The task is to determine the locations of the reflecting or scattering circles based on the detected total wave.

Summarized, compared to a fully general inverse scattering problem, we investigate a special situation. This can be characterized with the following:

- The number and shape of the objects are known: we take two identical circles.

- The structure of objects, i.e. their reflection properties, is known.
- The incident wave is known.
- The incident waves are harmonic, their frequency is constant.
- The reflected wave is known in a space-segment.

The problem described in this section has been studied experimentally for a long time; it is also the basis of different diffracting structural analysis procedures. This serves as the principle of geological structural investigations. Medical imaging methods, the physical basis of radar measurements, and the mathematical description discussed below are also related to the problem we study here. In a number of real situations, we can only detect the reflected waves from one side or only in a small part. A typical example is given by the measurements in geophysics, where waves are generated at the surface and the reflected ones are measured only at some points of the surface. In nature, bats' brains solves a similar but even more difficult task: they measure the reflected waves only at one point and process them in real time.

2.2 Description of the mathematical model

The following notations are used to describe the mathematical model:

- Ω^- is a union of open spheres,
- $\partial\Omega^-$ the surface of the open spheres,
- k is the frequency of the incident wave,
- $u : \mathbb{R}^2 \to \mathbb{C}$ is the complex amplitude,
- u^{inc} is the complex amplitude of the incident wave at the surface of the spheres.

As described in the previous section, the reflection and scattering of the wave is described by the Helmholtz equation:

$$(\Delta + k^2)u(\mathbf{x}) = 0, \qquad \text{if} \quad \mathbf{x} \in \mathbb{R}^2 \setminus \overline{\Omega^-}, \tag{2.2.1}$$

where $u : \mathbb{R}^2 \to \mathbb{C}$ is the unknown function. The absolute value of this gives the amplitude of the actual wave, and the direction of its trigonometric shape is the phase of the wave. A detailed description about electromagnetic waves can be found in the books [9] and [10].

To have a unique solution of the equation (2.2.1), this should be completed with some boundary condition (Dirichlet, Neumann, or mixed) according to the physical problem. A possible common choice is the following:

$$\begin{cases} (\Delta + k^2)u(\mathbf{x}) = 0, & \text{for} \quad \mathbf{x} \in \mathbb{R}^2 \setminus \overline{\Omega^-} \\ u(\mathbf{x}) = -u^{inc}, & \text{for} \quad \mathbf{x} \in \partial\Omega^-, \\ \lim\limits_{|\mathbf{x}|\to\infty} |\mathbf{x}|^{\frac{d-1}{2}} \left(\partial_r u(\mathbf{x}) - iku(\mathbf{x})\right) = 0, \end{cases} \tag{2.2.2}$$

where a Dirichlet boundary condition is given at the edge of the reflective object. It is important that the solution is well defined, hence a boundary condition in the infinity is also required. This is almost always the Sommerfeld radiation condition, written in the third line of (2.2.2). Here, a derivative taken as a function of the distance from a fixed point is used.

Remarks: 1. A more realistic assumption is to apply a Neumann boundary condition when instead of the second line of (2.2.2), $-\partial_{\nu(\mathbf{x})}u(\mathbf{x}) = -\partial_{\nu(\mathbf{x})}u^{inc}$ is satisfied for all $\mathbf{x} \in \partial\Omega^-$, where $\nu(\mathbf{x})$ is normal vector pointing outward from the object's surface $\mathbf{x}$.

This is mainly relevant in cases where the object completely reflects the wave. The homogeneous Dirichlet boundary condition, mentioned in the original equation is satisfied only in special cases. An example for this is the study of underwater sound waves, where the boundary is the water surface [6].

2. Even in this form, the discretisation and numerical solution of the equation is hopeless, since it is given in an unbounded domain.

Let G be the Green function for the operator $\Delta + k^2$. Then the integral representation of the solution of (2.2.2) is as follows:

$$u(\mathbf{x}) = \int_{\partial\Omega^-} G(\mathbf{x}, \mathbf{y})\rho(\mathbf{y})\,\mathrm{d}\mathbf{y}, \qquad \forall \mathbf{x} \in \mathbb{R} \setminus \overline{\Omega^-}, \tag{2.2.3}$$

where a $\rho : \partial\Omega^- \to \mathbb{C}$ density can be derived from the following integral equation:

$$-u_{\mathrm{inc}}(\mathbf{x}) = \int_{\partial\Omega^-} G(\mathbf{x}, \mathbf{y})\rho(\mathbf{y})\,\mathrm{d}\mathbf{y}, \qquad \forall \mathbf{x} \in \mathbb{R} \setminus \overline{\Omega^-}. \tag{2.2.4}$$

Fortunately, the dimension has been reduced by one in (2.2.4), which speeds up the calculation of ρ. Knowing this, the problem in (2.2.3) simplifies to the computation of a boundary integral, where we can approximate u for a given ρ at any point in the range of $\mathbf{x}$.

In our case, however, we do not know the object, such that here $\partial\Omega^-$ is unknown. Instead, we know the solution u itself, but only at a few points.

Accordingly, the problem to be solved is called an *inverse problem* [2] for equation (2.2.2), where for a known solution (denote this by $\tilde{u}$) is the domain we are looking for (Ω^-) is unknown.

Unfortunately, this task is not well-posed: neither the existence of the solution nor its continuous dependence on the measurement data can be guaranteed. As a result, the noise in the measurement data and the computational error in the numerics can highly increase the inaccuracy of any solution algorithm.

To illustrate this, let's take a look at figure 2.2.1 and 2.2.2! On the left, two different cases of the reflective objects are given, while the figures to the right show the average deflection of the reflected wave. It is clear how similar they are, despite the very different arrangements. Therefore, it seems obvious how difficult it is to predict the positions of the reflective objects from the data obtained during reflection.

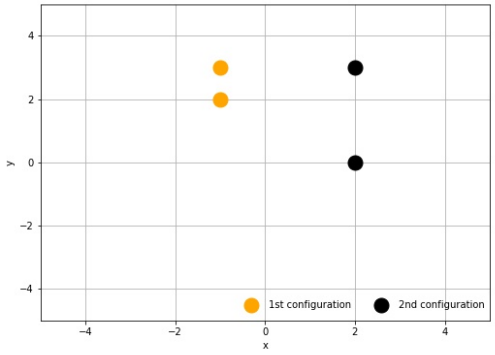


Figure 2.2.1: Two different locations of the centres of the reflecting objects in a case showing instability (yellow: first case, black: second case).

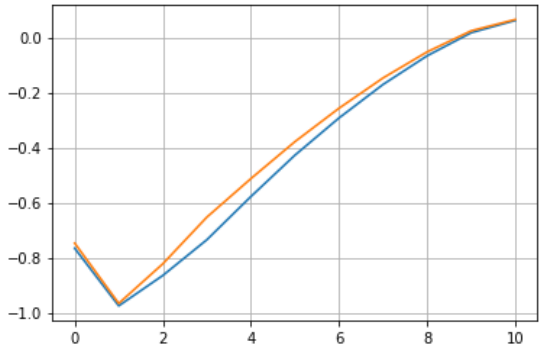


Figure 2.2.2: The image obtained by averaging the reflected waves. The vertical axis shows the deflection, the horizontal shows the location of the measurement (yellow: first case, blue: second case).

The basis of the conventional numerical approach is the following. For an arbitrary domain $\hat{\Omega}$, we denote the solution of the direct problem in (2.2.2) by $\hat{u}$. With this, as a first try, one should find the domain such that $\|\hat{u} - u\|$ is minimal. Here, a meaningful norm $\|\cdot\|$ should be defined. Indeed, this is still not satisfactory: usually, an extra regularising term should included in the minimization procedure.

Usually, the so-called the Tikhonov regularisation [3] is applied, which has an extensive literature.

2.3 Literature review

Solving inverse problems with analytical formulas is practically hopeless, such that in any case, one has to apply some numerical approximation. As mentioned above, this raises still many problems and can be very time-consuming as we have to solve a number of direct problems. Therefore, several authors have tried using neural networks to solve inverse problems in the recent years.

An exhausting review on these approaches would be a hard task, since the recent results are usually not published in journals, but rather on ArXiv, in institutional reports, or just as conference papers. Also, papers published in conventional journals are quite scattered - these can be found in physical, engineering, or sometimes mathematical journals.
Therefore, we have picked three sub-topics: one somewhat simpler and two even more complex problems from which several publications have been published.

We also note that the majority of the methods for detecting the position of some object are based only on reflection times. This can predict the position accurately but does not deliver sufficient information on the material properties of the objects and the surrounding medium.

2.3.1 Acoustic application

In this sub-topic an important application is to detect the source of an acoustic (sound) wave. In most cases studied in the literature, only have one source is supposed. This is a substantially easier problem compared to our case. In many cases, also the presence of noise is modelled and reflective walls assumed on the boundary. A brief summary of the topic can be found in the [11] work and a well-documented case is described in the [16] work. The domain is known here, and such procedures are based on a large number of training data sets.

2.3.2 Application for medical imaging

Due to the practical importance of this topic, much research has been done, although probably the most effective procedures and their methods are not published. As with practical problems in general, this is very difficult in several ways. On the one hand, the structure of the region to be examined is unknown; this is what needs to be explored.

It is also extremely hard to collect a training data set in this field, since we make a single measurement and already want the results. However, NNs must be trained in all cases, so the instruction must be on a general object. A detailed description of a specific task can be found in [5], and the journal *IEEE Transactions on Medical Imaging* is dealing with this problem as well.

It must be noted that due to the above difficulties, the use of neural networks for medical imaging involves the enhancement of existing, known images.

2.3.3 Application for geological structure analysis

Scattering problems are also of interest in geology and geophysics, which are also main fields of practical applications. The location of individual mineral resources (especially oil and natural gases) can be deduced from the way in which they conduct and reflect individual vibrational or seismic waves. These waves are usually generated by blasting.

Whenever the mathematical model of this phenomena has been known for a long time, its the numerical solution is very difficult due to the large amount of noisy data and the lack of well-posedness. Instead of these "direct" methods, neural networks can offer an automatised data processing by extracting information. [13] provides a modern overview of this topic with several additional references.

Chapter 3

The solution using neural networks

3.1 Neural networks in a nutshell

The neural network, for which the notation NN is used hereafter, can be given as a function. For the exact description we use the interpretation

$$\mathcal{NN} : \mathbb{R}^{N_0} \to \mathbb{R}^{N_K},$$

where N_0 is the size of the input, N_K is the size of the output, and K is the number of layers, which will be described later. Both the input and the output are often positive integers, i.e. formally: $\mathcal{R}_{\mathcal{NN}} \subset \mathbb{N}$.

3.1.1 Example. *A neural network that recognises ten digits.*
We choose the input as a grayscale image of size 200×200*, and its output should be the ten possible digits. This can be described by the following choice above:*

$$N_0 = 200 \cdot 200, \quad N_K = 10, \quad \mathcal{R}_{\mathcal{NN}} = \{0, 1, \dots 9\}.$$

In this case, the function $\mathcal{NN}$ is the following composition:

$$\mathcal{NN}(\mathbf{x}) = f_K(f_{K-1} \dots f_2(f_1(\mathbf{x})) \dots),$$

where

$$f_j : \mathbb{R}^{N_{j-1}} \to \mathbb{R}^{N_j}, f_j(\mathbf{x}) = \sigma_j(A_j\mathbf{x} + \mathbf{b}_j)$$

for some $A_j \in \mathbb{R}^{N_j \times N_{j-1}}$, $\mathbf{b}_j \in \mathbb{R}^{N_j}$ and $\sigma_j : \mathbb{R} \to \mathbb{R}$ specific functions that we use per each component. From a mathematical point of view, these so-called *activation functions* are essential building blocks of the network because they ensure that any nonlinear process can be simulated. These activation functions are present in the real neural networks as well: here an electrical signal does not propagate directly between neurons, only when it has reached a certain threshold. Based on this motivation, various functions can be used as σ_j.

The application of the function f_j is called the jth level or layer. In the last layer, usually $\sigma_{N_k}(x) = x$, i.e. the identical function is used.

As an example, we present a general three-layer NN . Here the input is the vector $\mathbf{x}$.

INPUT $\mathbf{x}$

1. LAYER $A_1\mathbf{x} + \mathbf{b}_1 \longrightarrow \sigma_1(A_1\mathbf{x} + \mathbf{b}_1)$

2. LAYER $A_2(\sigma_1(A_1\mathbf{x} + \mathbf{b}_1)) + \mathbf{b}_2 \longrightarrow \sigma_2\left[A_2(\sigma_1(A_1\mathbf{x} + \mathbf{b}_1)) + \mathbf{b}_2\right]$

3. LAYER - OUTPUT $A_3(\sigma_2\left[A_2(\sigma_1(A_1\mathbf{x} + \mathbf{b}_1)) + \mathbf{b}_2\right]) + \mathbf{b}_3$

For a more detailed description of the neurl networks, we refer to [4], [12].

As we have seen, a specific NN is determined by two basic compounds: how we choose the σ_j functions, and what the corresponding A_j matrices and $\mathbf{b}_j$ vectors are. In the specific definition of the NN, we fix the functions σ_j, and try to optimise the *parameters* A_j and $\mathbf{b}_j$.

This process is called the *training* of the neural network, which mathematically corresponds to an extreme value problem. To do this, we give a couple of known (observed or simulated) input-output pairs. We are looking for parameters which deliver the smallest possible deviation between the correct output and the result given by the NN. Here we also should use a metric to define a deviation between two outputs.

Considering the above general framework, the neural network can be specified exactly as follows:

- The σ_j functions used in each step must be recorded.
- The number of layers, the A_j matrix and b_j vectors must be specified.
- The metrics to use during optimisation must be specified.
- The optimisation algorithm itself and the associated parameters must be specified.
- The amount of iterations should be recorded.

In the first four cases, it is customary to choose from standardised cases. In fact, it corresponds to matrices with special structure transformations. The Keras package / library system for the Python programming language includes all of this with detailed documentation. For a detailed description of the language and its application to the implementation of NNs, see [17].

The case of our specific problem will be described in section 5.1.

In summary, we expect the neural network to predict the positions of the reflecting objects accurately, only based on the "experience", without any kind of solution formula or mathematical knowledge of the physical model. This is what we are trying to achieve when training the NN.

We get the data set as a result of the simulations, in which each row is a sequence of vectors. Formally, our raw data set has a shape of

$$\begin{pmatrix} \mathbf{v}_1 & \mathbf{w}_1 \\ \mathbf{v}_2 & \mathbf{w}_2 \\ \vdots & \vdots \\ \mathbf{v}_{M_{\text{opt}}} & \mathbf{w}_{M_{\text{opt}}} \\ \mathbf{v}_{M_{\text{opt}}+1} & \mathbf{w}_{M_{\text{opt}}+1} \\ \vdots & \vdots \\ \mathbf{v}_M & \mathbf{w}_M \end{pmatrix}, \tag{3.1.1}$$

where M is the number of attempts, and for each possible j index $\mathbf{v}_j \in \mathbb{R}^{N_K}$ and $\mathbf{w}_j \in \mathbb{R}^{N_0}$.

In our case $\mathbf{v}_j$ corresponds to the objects, while $\mathbf{w}_j$ corresponds to the reflected wave (for example, to its absolute value). We train the neural network with the first values of number M_{opt}. For an effective training procedure, a relatively large data set must be provided.

This is divided into two parts: one part is called the training data set, which is used to optimise the parameters. This is approx. 90 to 95 percent of the whole data set. The other set of data is used to verify that the NN has succeeded in optimising the parameters and results in an accurate output. In these cases, we can compare the output given by the neural network with the real values. Accordingly, this set of data is called the testing data set.

Formally, we are looking for a NN, for which the approximation $\mathcal{NN}(\mathbf{w}_j) \approx \mathbf{v}_j$ is as accurate as possible for $1 \leq j \leq M_{opt}$.

3.2 Training and testing in more details

In case of $j \leq M_{\text{opt}}$, $\mathbf{v}_j$ is compared with the known result. Some optimisation algorithm is used then to find an optimal parameter set in the NN. The task performed is the following:

$$\sum_{j=1}^{M_{\text{opt}}} \|\mathcal{NN}(\mathbf{w}_j) - \mathbf{v}_j\|^2 \longrightarrow \text{MIN}, \tag{3.2.2}$$

where $\|\cdot\|^2$ denotes the square of some norm. The left-hand side is called the training loss.

The accuracy of the result is verified then by running the neural network with a different data set. In concrete terms, we use the data pairs $(v, jw_,)$ with $M_{\text{opt}} < j \leq M$, the testing data set. In concrete terms, we calculate

$$\sum_{j=M_{\text{opt}}+1}^{M} \|\mathcal{NN}(\mathbf{w}_j) - \mathbf{v}_j\|^2, \tag{3.2.3}$$

which is called the validation loss. From the magnitude of the loss function, we can deduce the accuracy of the NN, i.e. how accurately this can predict the real output.

The function to be minimised can be defined in different ways depending on the underlying metrics. The Keras library offers a number of possibilities.

In a more detailed analysis of the optimisation, we also monitor the change of the loss function in the formula (3.2.3) and the validation loss function in the formula (3.2.2) obtained during the training.

Accordingly, the Python code uses a separate variable for the error (3.2.2) received when training the NN and the (3.2.3) error received during testing. In practice, due to the large sample numbers, the average of these quantities is displayed when the program is running. The phenomenon of overfitting can also can be detected using these two data: If the value of (3.2.2) is no longer decreasing when running the program, while the value of (3.2.3) received during testing is starting to increase, overfitting occurs.

3.3 Principles during simulation

Consider the physical problem described in chapter 2. We measure the reflected waves from a rectangle-shaped medium, with non-overlapping, unknown objects inside and intend to estimate the locations of the circles.

In order to obtain a sufficiently large set of training data set, simulations with many possible geometric arrangements must be performed. For simplicity, we assume that there are two reflective objects, each of which a circle with a unit radius. The result of the jth simulation, which belongs to the input of the neural network, is the absolute value of the reflected wave, denoted by $\mathbf{w}_j$. It will be shown later, that this needs to be transformed before we actually define the first layer.

To determine the structure of the NN, we need to take into account the physical and mathematical nature of the problem. Although attempts have been made in the literature to perform a mathematical analysis of the operation of a neural network, tailoring it to a specific task is not a common practice at all. Specific details are given in section 5.1.

When defining the different layers in the NN, it's extremely important to take into account the phenomenon of overfitting mentioned above, which is the greatest danger during the operation of the NN. To understand this problem, take a training set of 500 vectors with the output of the NN being a single number. The training data set should be given by the first 450 vectors. Fitting a polynomial of degree 450 is completely accurate with zero errors. However, it is almost certain that for the other 50 test data this polynomial would give an extremely large error. The problem is that we use too many parameters. Although we got an exact fit for the training data set, it became

very specific to it, which was achieved at the expense of the accuracy of the solution on arbitrary data. This must be avoided at all cost.

Chapter 4

Simulation for creating the training and testing data sets

For the simulation, we used a Matlab program package called μ `-diff`, developed by Xavier Antoine and Bertrand Thierry in 2014 at the University of Lorraine [15].

Our goal with the program is that after placing the scattering centres on a grid of size $n \cdot n$ and solving the scattering equations with the program, we obtain data to train the neural network. To do this, the data we assign to each other is the radius of the objects and the coordinates of their centres (v in equation 3.1.1), from which the scattering amplitude must be determined.

4.1 Description of the program

The *mu* `-diff` package expects the following parameters as the input:

- grid size (the size of the rectangle)
- number of scattering objects
- radius and centre for each of the scattering objects
- wave number
- for point sources: location, for plane waves: angle of incidence

With these inputs, the program performs the necessary calculations: it determines the reflected wave over the entire range.
The output is the value of the reflected wave measured at the bottom of the rectangle. This will be referred to as *reflected wave*. This corresponds to real situations (radar, sonar, ultrasound) when the reflected signal is detected on a relatively small surface. Only knowing this would make a traditional simulation very difficult, but we expect the neural network algorithm to successfully handle this case as well.

Once we received the reflected wave, it was transformed to make it easier for the NN to work with it. This corresponds to w in equation 3.1.1. We found that NN could not use the strongly oscillating data: the loss function hardly decreased - the optimisation

practically did not work. For this reason, the data were transformed to be "smoother" - with less oscillation.

1. First, we determined the local extreme values and their locations.
2. Then we took the average of the extreme values and associated with the average observation points. Consequently, we got as many pairs, as many waves were observed.
3. The data were then linearly interpolated between the above taken values, giving us the average deviation of oscillations.

This is illustrated in figure 4.1.1. We can say that we obtained a linear function with as many break points as there were periods in the original perceived signal. Despite the data loss, this method significantly improved the results obtained by the NN.

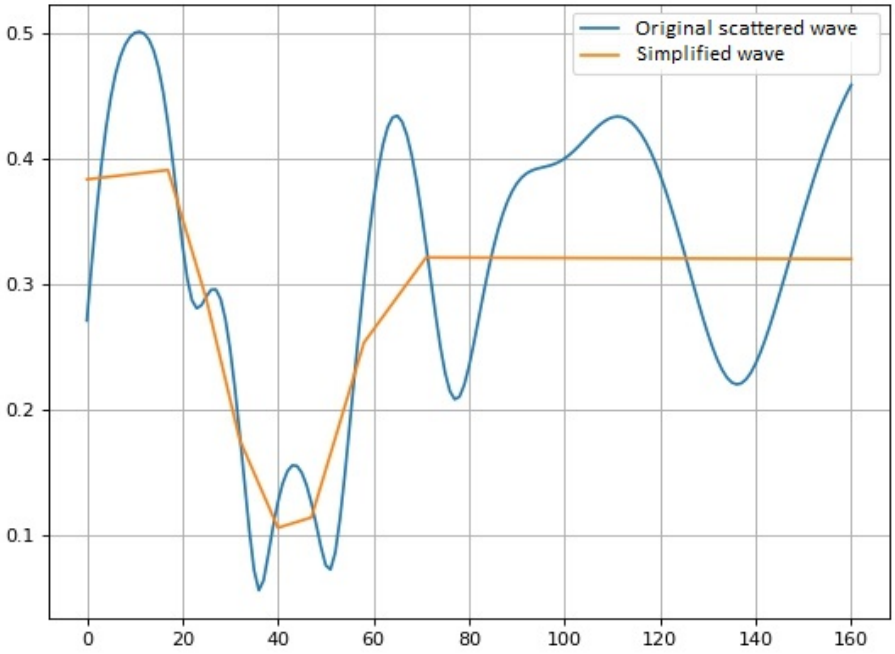


Figure 4.1.1: The first transformation of the reflected waves.

4.2 Simulation results

The simulation results are described when the centres are located on the integer internal grid points of the rectangle $(-5;5) \times (-5;5)$. Taking all such cases into account, we perform the simulation.

Each case is specified exactly by $\mathbf{v} = [x_1, x_2, y_1, y_2]$ vector, where the x and y coordinates of the centres are listed. It is important to consider $\mathbf{v} = [x_2, x_1, y_2, y_1]$ to the one mentioned above. It's not only important, because it reduces the computation time,

but also because otherwise we would get two minima during the optimisation process. This would make the operation of the algorithm unreliable and more difficult.

Taking the above procedure, we run the simulation for 735 cases, considering the possible different geometric situations. This took only a few minutes, depending on the computing capacity of the used PC.

Some results of two such simulations are illustrated in the following figures.

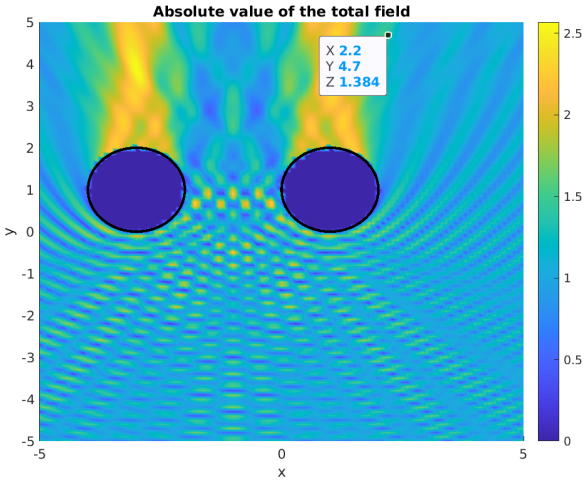


Figure 4.2.2: Absolute value of the total (detectable) wave obtained after reflection in the rectangle medium for spherical reflecting centres with coordinates of $(-3;1)$ and $(1;1)$.

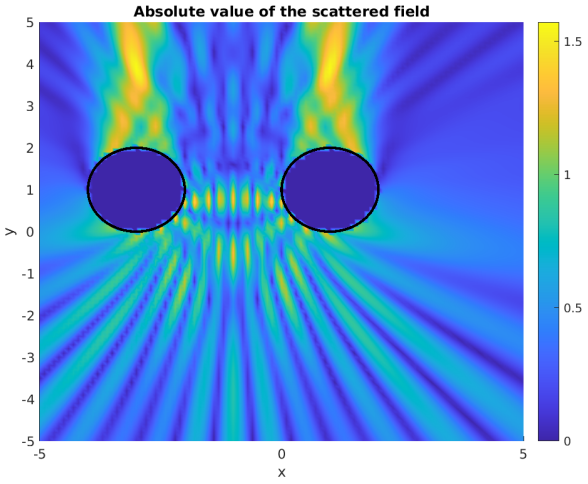


Figure 4.2.3: Absolute value of the reflected component of the wave obtained after reflection in the rectangle for spherical reflecting centres with coordinates of $(-3;1)$ and $(1;1)$.

In both cases, a vertically upward plane wave with a wavelength of $4 * \pi$ was used. The waves reflected from the scattering objects were detected at the lower edge of the rectangle medium. We use the reflected component of the wave for further analysis to determine the positions of the objects.

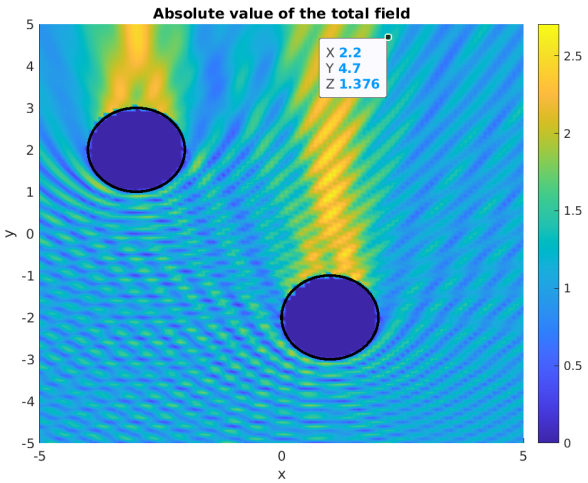


Figure 4.2.4: Absolute value of the total (detectable) wave obtained after reflection in the rectangle medium for spherical reflecting centres with coordinates of $(-3;1)$ and $(2;-2)$.

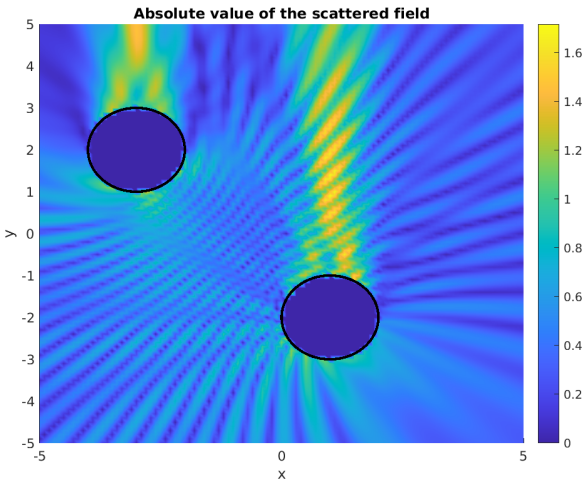


Figure 4.2.5: Absolute value of the reflected component of the wave obtained after reflection in the rectangle for spherical reflecting centres with coordinates of $(-3;1)$ and $(2;-2)$.

The value of the reflected wave was obtained at 160 points, and so was the transformed vector.

Both the calculation results and the scattering object coordinates were collected in a `txt` file. Since we want to obtain a large data set for training the NN, we irradiated the scattering centres from several sides, by changing the angle of incidence, and then used the resulting reflected waves. We first examined the angles of $[n \times \frac{\pi}{2}, n \in \{0;7\}]$. This represents a total of 8 *times* 735 reflected waves. Before using this, it must be normalised, so the efficiency of NN operation is further improved. Finally, the values are divided into testing and training data sets.

It was observed, that several produced configurations of the scattering centres were similar. These occur mainly when the spheres are "behind each other" (covered). This is illustrated in figure 4.2.6.

It is clear that the resulting reflected waves are almost identical. When working with such data, it is very difficult for the neural network to distinguish these scattering centres, so we usually get a larger-than-average error in similar configurations.

There is also an interesting case where two completely different configurations lead to a similar reflected wave. These could possibly be eliminated by taking all the smallest distances between the reflected wave vectors and then checking for the smallest ones that the distance between the points themselves exceeds a threshold. If so, we eliminate both data so that neither can interfere in any way. If, on the other hand, their distance does not exceed a predetermined value, we do not need to do anything, as nearby points are expected to give a similar reflected wave. However, the selection of these points in the present situation goes beyond the scope of this research.

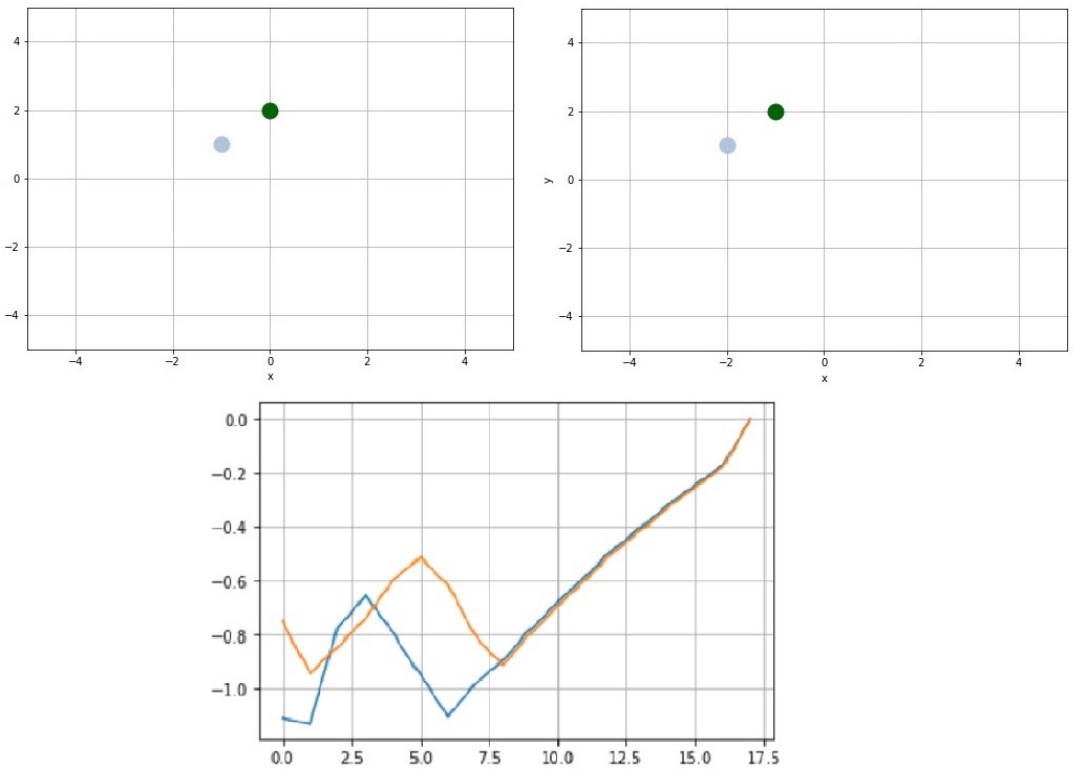


Figure 4.2.6: Representation of the reflected waves of two different configurations (bottom) and the locations of the two scattering centres (top left and right).

4.3 Transformation of the data

When the wavelengths are comparable to the size of the objects, we obtained strongly oscillating solutions with which the optimisation algorithm did not work, i.e. the loss did not decrease significantly during each optimisation step.

Therefore, data transformation is required.

- First, we define the magnitude of the amplitude at each division point. For this, the local extremes of the data set were determined, and then the amplitude was defined as the difference between the adjacent local minimum and maximum at each division point.
- Second, we calculate the distance between the locations of the local extremes and define this as the local wavelength between the locations. This will actually be half of the estimated wavelength.

These transformations also have physical motivations. Sound wave amplitudes indicate the strength of the reflected sound and the wavelength indicates the pitch of the sound. We therefore want to infer the position of the reflected objects by observing the strength and height of the reflected sound at quite a few points.

It is important for simulations with neural networks that the lengths of the vectors used as the input is the same. For this reason, the amplitude and wavelength must be estimated at each location, even if this is technically more difficult.

All this was done in the Matlab software package. We tried to create efficient program code to perform the above task. There is a detailed documentation, so one can easily follow through the steps when reproducing the work.

The original- and the transformed data sets for a specific example are illustrated in the following figures.

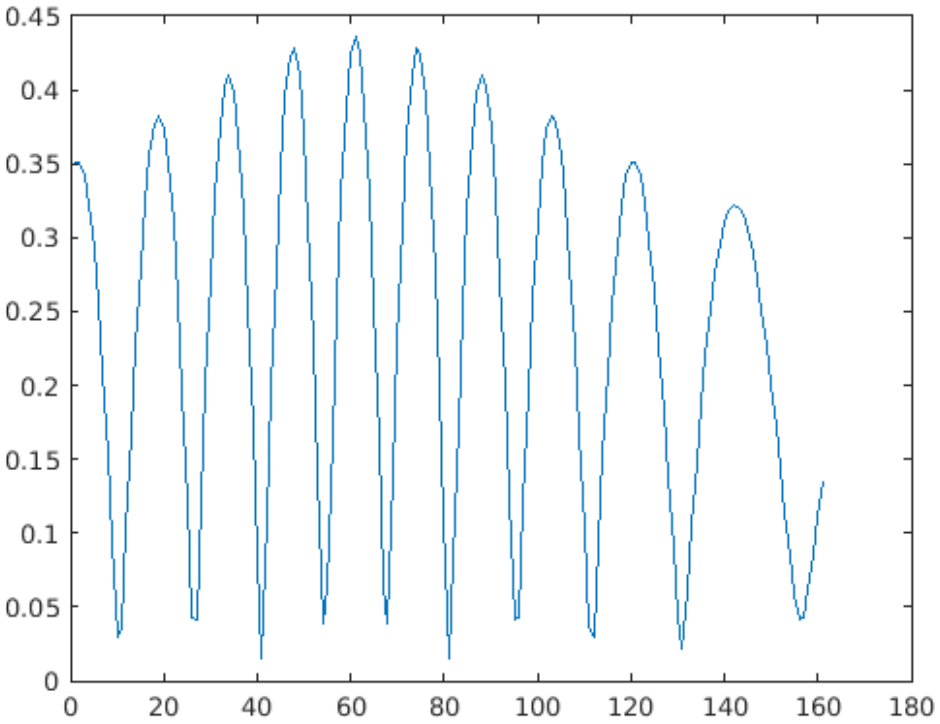


Figure 4.3.7: The data set obtained by the simulation, where the wavelength of the vertically propagating wave and the wave reflected by the objects was calculated at 160 grid points.

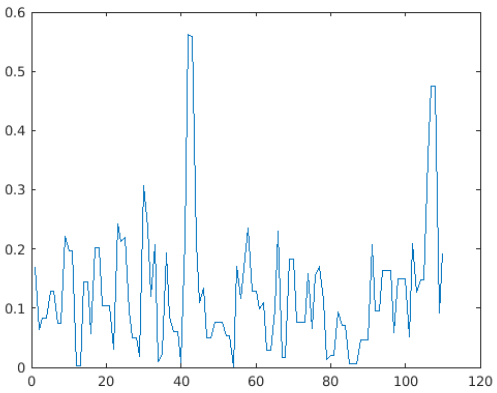


Figure 4.3.8: The amplitudes calculated at each location in the data set obtained by the simulation.

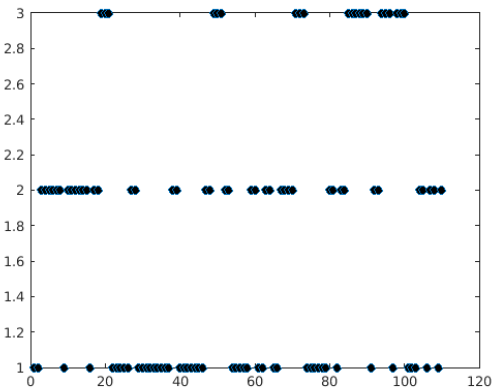


Figure 4.3.9: The wavelengths calculated at each location in the simulated data set.

After the transformation, in order to increase the efficiency of NN, a so-called normalisation procedure is also needed, which transforms all the components of the input vector into the interval $[0, 1]$. Often the interval $[-1, 1]$ is chosen for this purpose. We normalised the data by a simple division but there are more complex procedures as well.

Other than applying this procedure, noise can also be filtered out, which has a strong negative effect on many real observations. Even though the data set we created for this simulation is noiseless, we added Gaussian nose to all our training- and testing data, so the research is more realistic. The amplitude of this noise was one-fifth of the standard deviation of the original plane wave's standard deviation. An example of this can be seen in figure 4.3.10.

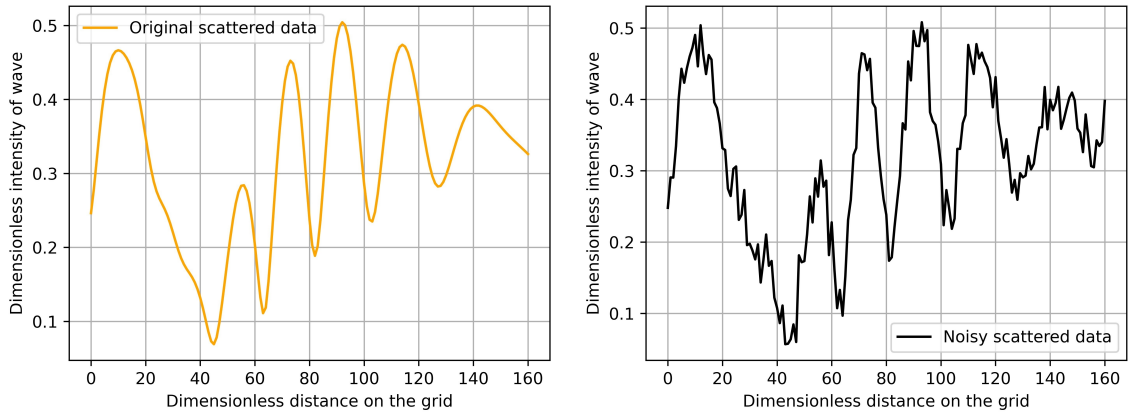


Figure 4.3.10: A sample simulated scattered data (left) and a corresponding noisy data (right).

After creating the noisy data, one can compare the preprocessed data set arising from the original, noiseless simulated data and the corresponding noisy data. This can be seen in figure 4.3.11. As It's seen, even though noise was added, the final preprocessed and normalised vector is almost identical to the non noisy one.

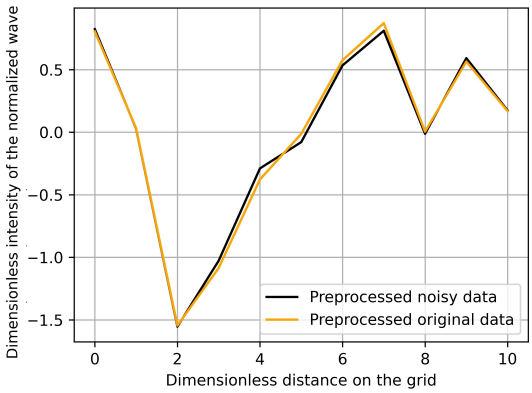


Figure 4.3.11: The original preprocessed data compared to the corresponding noisy data.

Chapter 5

Solution obtained with the neural network

In this chapter, we describe in details the NN constructed for the inverse scattering problem. We will summarise our experience gained during the computations, and finally the results of the various simulations will be presented and discussed.

5.1 The structure of the NN

As mentioned, constructing a feasible NN for for simulating a real-life phenomenon is the most difficult task in this topic. Likewise, it is also the most important task, as the efficiency of the simulation procedure (i.e., the accuracy of the corresponding numerical solution) depends largely on this.

There are no general recipes for constructing appropriate NNs. Therefore, this often consists of fine-tuning a standard mesh structure, based on some common procedure. For example, the basic structure of NNs used for image processing are very similar, most of them can be considered some mutations of each other.

Although, we could also not pre-determine an optimal structure of the network to be used, we will mention several aspects that determined the construction of NN.

We have first considered NNs for similar purposes: in the literature several works are available for detecting sound sources. In both [11] and [5] works, NNs were constructed that used convolutional methods in the initial layers, and then dense layers. It is important to find specific patterns or structures in a localisation problem. For this purpose, multiple convolutional layers were used, including local maximum-calculating and averaging layers. We started from this construction, which we modified in several ways according to the problem under study. In each case, we followed the principle that if something is known about the data, that knowledge should be incorporated into the structure of the NN.

To illustrate the structure and performance, a summary table was obtained when running the program. The program itself is given in the appendix of the dissertation. In this, not only the procedure used was presented, but also some attempts leading to it can be traced.

To understand the descriptions in the following sections, see section 5.1.4.

5.1.1 The layers

The first task is to choose the number of layers and their types in the NN. The corresponding numbers and sizes are the result of consecutive experiments.

First hidden layer

The input layer is given, its size is determined by the input data set. Recall that this is the reflected wave data, which we already transformed.

It is important to specify the structure of the input layer by treating those input data separately which were reflected from plane waves with different propagation angles. In this way, we incorporate the information that these are derived from separate observations.

Of course, the question arises how to group observation data. Could it be more appropriate to collect all data in a fixed observation point, and collect observation data in the next point and so on. We have tried also this data structure, but the networks based on this method proved to be less efficient. This is interesting, since the data obtained during image recognition is processed per pixels, i.e. each pixel has a vector that specifies the colour scale and brightness.

Based on the above description, the input layer contained 11x16 long vectors, where 16 refers to the direction of the plane waves and 11 to the length of the data in the processed data set.

The first hidden layer

We want to obtain information from each reflected wave. Accordingly, for each of them convolution layers are applied. This is usually associated with the fact that observations usually have several significant properties, which can be quantified by well-chosen coefficients in convolutions. Since we are looking for several such features, we took 12 different convolution coefficient vectors for each observation.

Since the reflected plane waves from different directions have different shapes due to the reflection, it is possible to pinpoint the positions of the searched objects based on their different characteristics. For this reason, each of the coefficients determining the convolutions were chosen to be different in the 16 observations. The convolution was thus one-dimensional, with a convolution kernel consisting of four elements. The latter is also called the convolution window. This is then convolved with the 11-element input vectors, which yields an 8-element vector in each case.

Accordingly, the first hidden layer will contain 16 matrices with the shape of 12×8, where 16 refers to the direction of the plane waves, 12 to the searched feature numbers in each case, and 8 to the features themselves.

Second hidden layer

Here, the previously obtained data is simply collected into a vector. No unknown parameters or weights are used for this, so this step does not affect the complexity of the optimisation (or training) procedure. This layer thus consists of a single vector of length 1536.

Further layers

The rest of the layers are *dense* layers, which means that each of the vectors are affected by each components of the previous layer. This is typical when looking for a particular pattern or trying to condense the information gathered in the previous vector into a smaller vector.

In our NN, the other such layers contained 50, 40, and 8 elements, respectively. We do not know any rule or method for determining the optimal sizes of these layers. For this reason, after a long trial and error, we obtained the final version, which works optimally according to our experience.

Output layer

The output layer contained the four coordinates of the centres of the unknown spheres. This was obtained from the values of the previous 8-element dense layer.

5.1.2 Number of parameters and its reduction

The application of multiple dense layers is simple, however, many parameters are generated during the application. For this reason, they should not be chosen too large. Moreover, it is possible that two selected points in two layers do not actually depend on each other. This seems to be the case in the training procedure, where the parameter describing their relationship does not change in the learning (optimisation) procedure.

In this beneficial to eliminate them, not only to speed up the algorithm but also to prevent the phenomenon of overfitting. That can be done by a *dropout* layer, where a certain proportion of the parameters with the smallest variation are eliminated between two layers [14]. Visually, it also seems most expedient to apply this to the first dense layer. In our network, half of the variables were eliminated here. If we had done this in a later layer, we would not have been able to omit as many variables and thus more useful connections would have been lost.

Generally, it is not advised to perform such an elimination at the convolutional steps. More dropout layers were experimented with in the latter dense layers, however, they did not improve the efficiency of the NN.

In concrete terms, summarising the above description, we have used the following number of parameters:

$$[(16 \cdot 8) \cdot (4 \cdot 1) + (16 \cdot 8 \cdot 1)] \cdot 12 + (16 \cdot 12 \cdot 8 + 1) \cdot 50 + \\ + (50 + 1) \cdot 40 + (40 + 1) \cdot 8 + (8 + 1) \cdot 4 = 86934.$$

5.1.3 The activation function

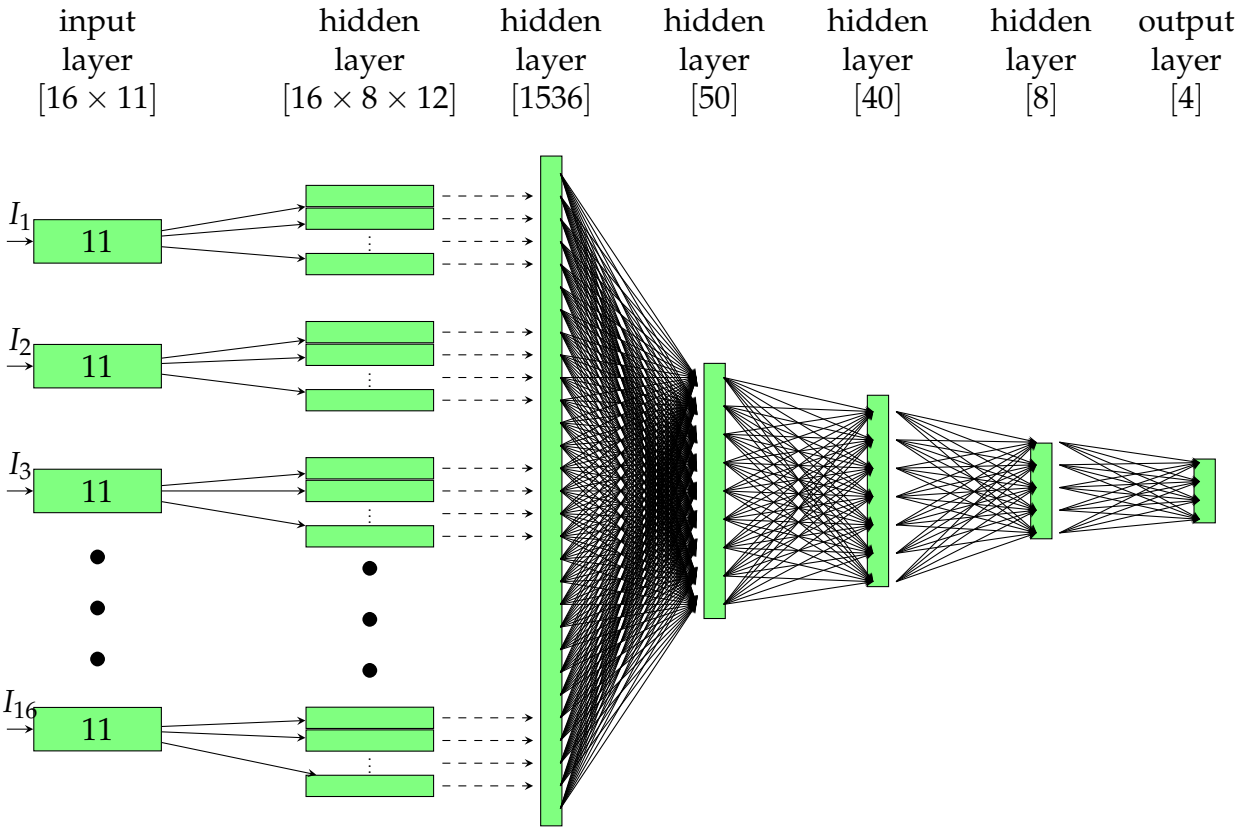
Since the values have a moderate upper bound and they are positive and also, they were smoothed with the preliminary data processing, `RELU` activation function was used. This is zero for negative values and identity for positive values. This might be a simple activation function, but it proved to be the most effective one at the same time. The results were very similar in the cases when the tangent hyperbolic `tanh` activation function was used. Interestingly, it is very common to use this activation function in theoretical studies of NNs [18].

In the case of several similar algorithms, it has been suggested that a so-called sigmoid activation function should be used in the last layer. This is confirmed by our computational experience as well.

5.1.4 Figure: The Neural Network

The above details are illustrated in the following figure. The individual layers and their connections are shown: one can distinguish the convolutional layers from the dense layers. The dashed arrows indicate that no new parameters or variables were used. The length of the hidden layers is not to scale, the largest would obviously not fit in the figure.

The dimensions of the input and the layers are also shown. When programming in Python, both an artificial dimension extension- and permutation is needed, so the desired operations can be performed by the program. This technical issue is not detailed in this dissertation.



5.1.5 Training the Neural Network

One can select the value to optimise when using a neural network: we chose the mean squared deviation. This means that the distance between the estimate given by NN and the known exact values is the sum of the squares of the circle centres determined by them and the true circle centres.

The optimisation method is some sort of stochastic gradient algorithm [1]. The performance and analysis of such procedures is a recent research topic.

We chose the `ADAM` optimisation algorithm [8]. Here, we also tried several learning rates and ended up taking a slightly less than the standard value. The instability in the inverse problem means that even a small change in the parameters entails a serious change in the results. For this reason, it may be appropriate to set them less than the standard rate in each step, as demonstrated by computational experience. Thus, our choice is a `learning_rate`=0.0005.
For the values of the other two parameters that determine the steps, the followings proved to be the most effective: `beta_1`=0.9, `beta_2`=0.8.

The operation of stochastic gradient algorithms can be split into large steps and smaller sub-steps. In each major step of the algorithm, we use all the training data, this is called *epoch*. In the smaller sub-steps, we use less data, which is called *batch*.

When creating the neural network, you can specify both the number of larger steps and how many data to use within them at one time. If a very low learning rate is chosen in the optimisation procedure, the number of epochs required to achieve a given accuracy will be much larger.

We must also take into account that the number of epochs can be chosen too large: then there is a danger that the phenomenon of overfitting will occur.

Based on our computational experience, for the present task, the optimal number of epochs is at about 100-150.

5.2 Results obtained with the neural network

When testing the neural network, the centres of the circles were placed on the grid points of the 5×5 rectangle; two congruent circles of unit radius. The task was to test, if the neural network could predict the coordinates of their centres based on the reflected waves.

The so-called *training data set* contained 676-, and the *testing data set* contained 59 vectors selected randomly from the entire data set of 735 vectors. This means that eight percent of the total data set was used only for testing, and the rest for training the neural network.

5.2.1 Effect of similar reflections

To examine this problem in detail, we calculated the reflected waves' distances, already mentioned in chapter 4.2. This can mainly be used to filter out problematic cases and thus making the neural network prediction more reliable. The distances of

the vectors is measured by the standard Euclidean distance, which is easily calculated by the subroutines of the *numpy* directory. The distribution of the distances amongst the captured signals is shown in figure 5.2.2. It is also clear from figure 5.2.2 that there are many pairs of waves (about 20,000) whose distance is rather small, which can make the prediction more difficult.

To analyse the problem in more details, we examine those geometric locations of the spheres, which led to the smallest distance of the reflected waves. If the distance between the spheres is very small, there is no problem, since in the case of similarly positioned spheres, we obviously get a similar signal. Recalling the figures 2.2.1 and 2.2.1, however, it is obvious that there will be spheres that are far apart, although we get a very similar reflected signal. A visual representation of such a case is shown in figure 5.2.1. These are called degenerate cases and it may be worth filtering them out of the training data set, as then the neural network would not be able to distinguish which data to assign to the received signal. This was done in the research, but interestingly, it did not significantly improve the efficiency of the procedure. Perhaps the reason for this is that we worked with less training data that way.

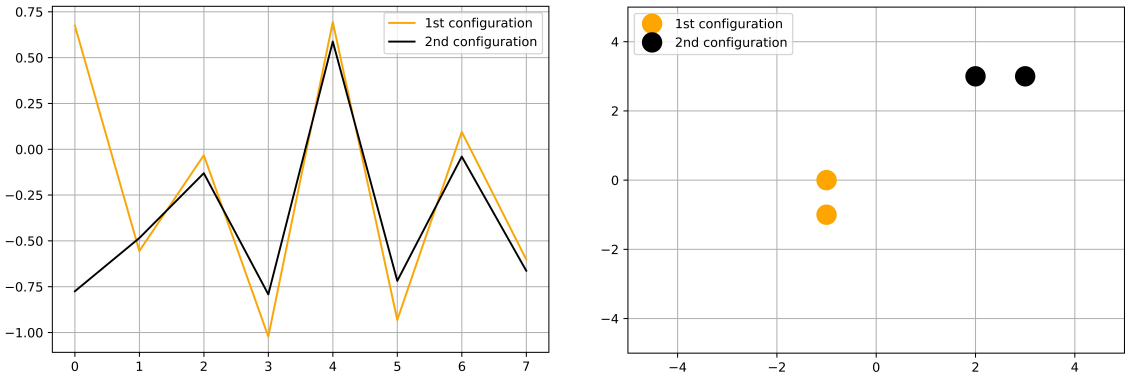


Figure 5.2.1: The positions of the waves (left), and the real geometric positions (right) in a degenerate case.

5.2.2 Efficiency of the neural network

The efficiency of NN is measured by tracking the decrease in the loss function on the data set used for testing. The loss function specifies how accurate the prediction given by the neural network is on the test data set, not used in the minimisation. We emphasise that the fit is done using the training data set, minimising the training loss function, and its accuracy is measured by the validation loss function.

These two quantities are illustrated in a typical case we get in figure 5.2.3. When running the program, these are considered important output values, and we often relied on them when tuning the network.

It can be stated that both loss functions decreased significantly in all cases, so the estimation to determine the centres proved to be efficient. However, the accuracy of the model could still be improved.

As it is seen in figure 5.2.3, first the validation loss function decreases together with the training loss function, then the decrease of the former stops or slows down, while

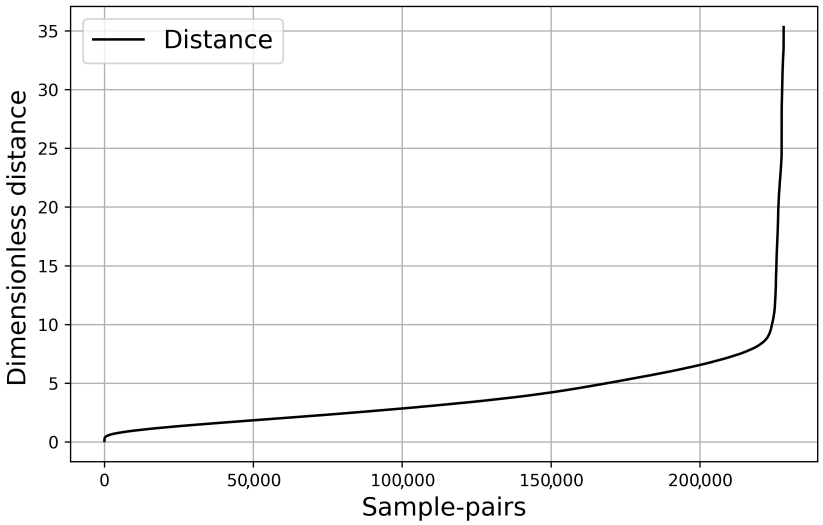


Figure 5.2.2: Distance distribution of the waves. xaxis: the number of sample-pairs with less distance than y

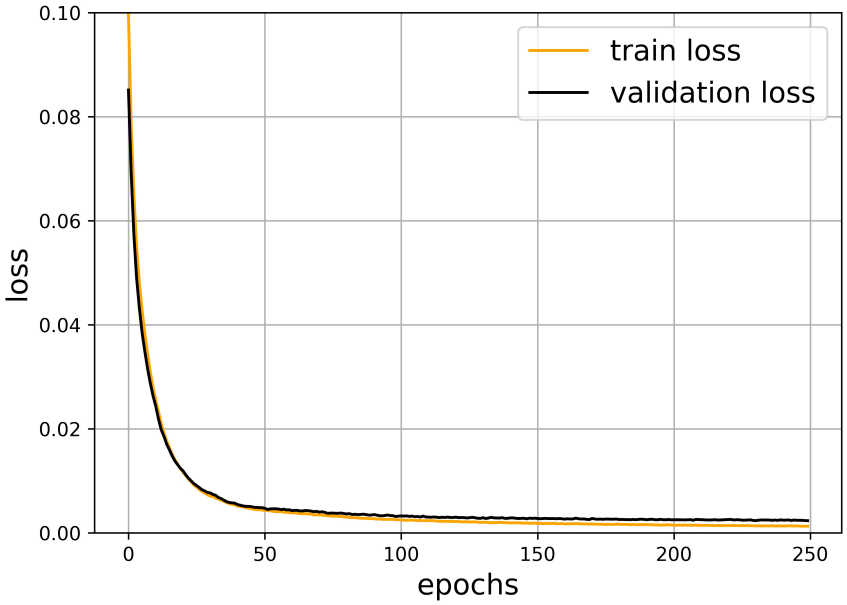


Figure 5.2.3: The training and validation loss function.

the latter continuously decreases further. Moreover, their value also oscillates, which is caused by attempts in different directions in the gradient method. It is usually sufficient to run the code until the two functions are separated, since the validation loss function is no longer significantly reduced after this. Accordingly, the network can be run approximately through 150 epochs; figure 5.2.3 is used to illustrate the long-term behaviour of loss functions.

As mentioned before, we attempted to investigate how the noisy data effects the sim-

ulations. Since the preprocessing method we used filtered out the noise in the used data set, the results were only slightly effected by this phenomena. We also tried to run the neural network without any kind of preprocessing, only on the noisy data set. This gave slightly worse results, for which the training and validation loss function can be seen in figure 5.2.4.

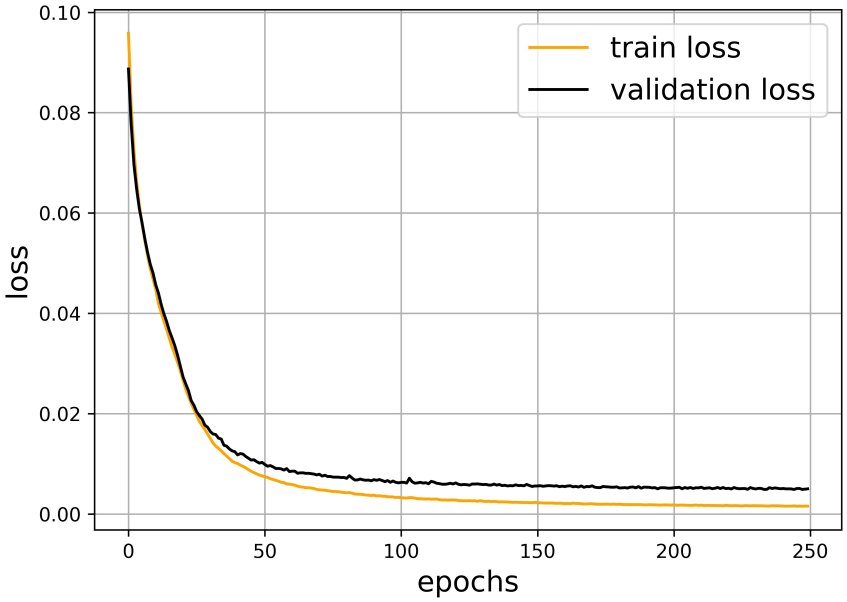


Figure 5.2.4: The training and validation loss function for the noisy and non-preprocessed data set.

When we didn't apply any preprocessing, the loss function remained two-three times the one in the original network. Moreover, since the size of the input data set had the dimensions of 16×161 instead of 16×11, the computational time was about ten times more.

To emphasise how important it was to use locally connected layers, we created a new NN, where only convolutional- and dense layers were used. The computational costs were about the same as in the original NN by this case. The results can be seen in figure 5.2.5.

As it's seen in figure 5.2.5, the validation loss function remained twice the original value. When using convolutional layers, the train loss remains above the validation loss, which suggest that new variables should be included. This is also a purpose of using the locally connected layers.

5.2.3 Presenting some results

One can observe in figure 5.2.6 and figure 5.2.7 that the neural network predicted he location centres of the spheres quite well using the data from the reflected waves. The left and right parts of the figures show spheres of different sizes. The larger size reflects the true size of the spheres, while the smaller one is included in the document only to visualise the difference between the real- and predicted centres.

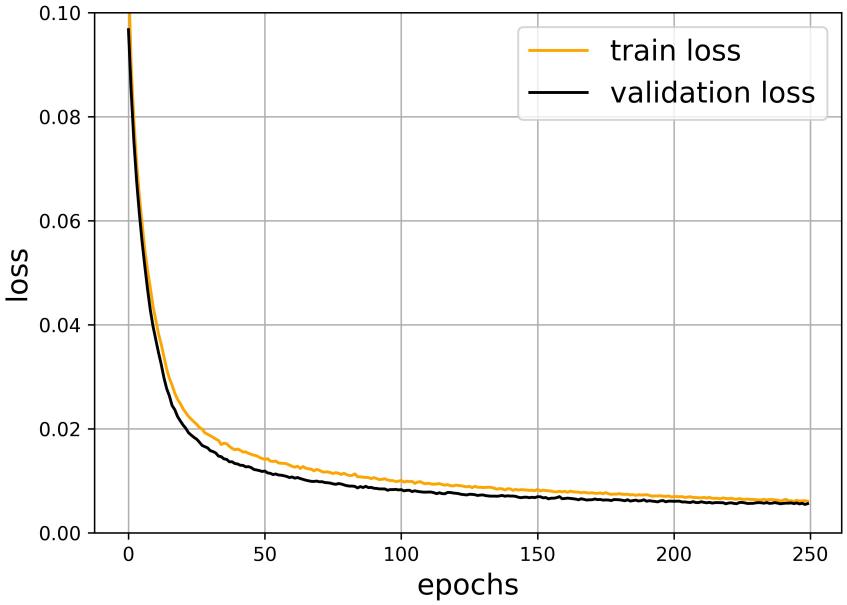


Figure 5.2.5: The training and validation loss function with convolutional NN.

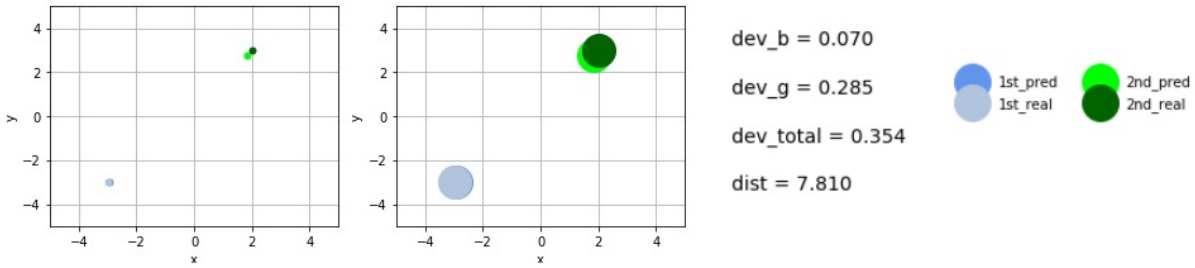


Figure 5.2.6: Example for the real and predicted centres.

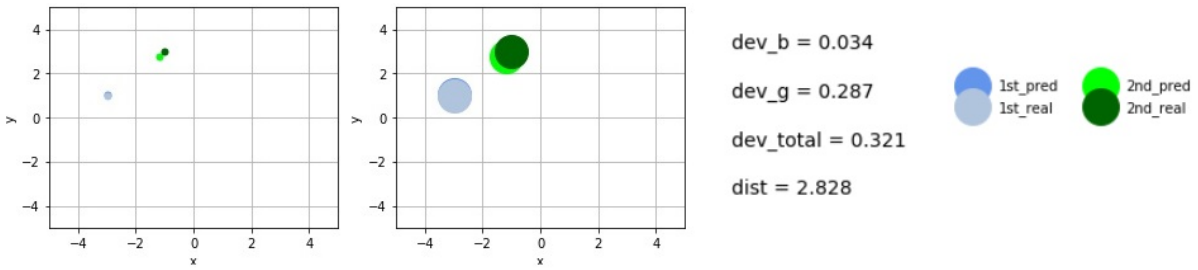


Figure 5.2.7: Example for the real and predicted centres.

When taking a closer look at the axes, one can observe the distances between the two sphere centers (`dev_b` and `dev_g`, where b refers to the blue- and g refers to the green spheres). Finally, the quantity `dev_total` is the sum of these, and `dist` is the distance between the centres of the spheres.

5.2.4 Error and the relationship of the positions of the spheres

For the experimental analysis of the computational algorithm, we wanted to answer the question of what are the "critical cases", i.e., when does the estimation of the centre of the spheres give a large error. Specifically, we examined the extent to which this depends on the distance from the centre of the spheres.

To do this, using all elements of the testing data set with all sphere configurations, we calculated the actual- and predicted locations of the centres and plotted it as a function of the distance between the two spheres. The results can be seen in figure 5.2.8, where the error refers to the distance between the predicted- and real centres. Here, the neural network ran over 150 epochs.

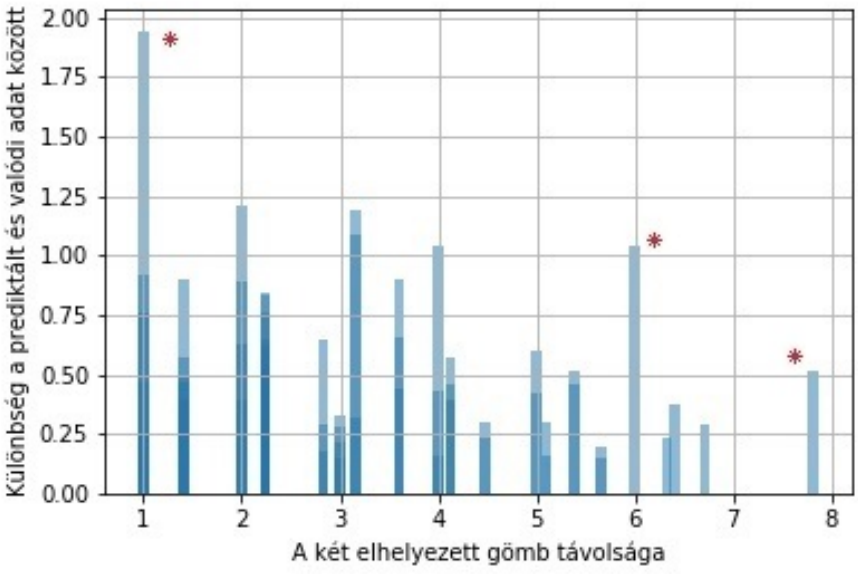


Figure 5.2.8: Error of the centre coordinates.

In the figure, we can observe, that with the exception of a few data, the magnitude of the errors decreases continuously as a function of the distance. This may be because if the spherical centres are very close to each other, they are difficult to distinguish from the reflected waves. However, it is well predicted that both spherical centres will be in a given smaller region, so overall the error will not be too large.

For the results marked with a red stars, there were few sample elements (i.e., there were few spheres in the test data that were just that far apart). These uncertain results can be omitted from the research.

It can be seen in the figure that even in the worst case, the result obtained from the neural network had a deviation of only the length of the radius of the sphere, which is an acceptable result. The range, mean, and standard deviation of our results presented in the unit of the grid:

- Coverage of the error: $r = 1.800$.
- Mean of the error: $m = 0.536$.
- Standard deviation of the error: $s = 0.334$.

After removing the three values, which had the highest error:

- Coverage of the error: $r = 1.064$.
- Mean of the error: $m = 0.502$.
- Standard deviation of the error: $s = 0.277$.

As it is seen, with the exclusion of points with the largest error, the coverage improved a lot, the mean and standard deviation not so much. Further improvement of these results discussed in chapter 7.

Chapter 6

Conclusion

In this research, a neural network based approach was presented for solving the inverse scattering problem with restricted observation of the scattered data. Including a complex locally connected layer in the NN seems to be the right choice; this ensures sufficient complexity, while a moderate number of parameters are used in the model. Another cornerstone in solving the mentioned problem was a feasible preprocessing of the data. We have assisted the network by pre-mining information, which resulted in a reduced number of parameters. This not only resulted in the speed of the simulations, but also made them more reliable by avoiding overfitting. Another great result of the preprocession was that the effect of noise was completely diminished.

This study provides a general framework, fixing that deep neural network with an appropriate preprocessing and locally connected layers are perfect for this job.

At the same time, the present work has its limitations: in a robust method, the number and shape of the scattering objects and their material properties are unknown. For a corresponding extension one should first detect the number of objects with a simple network and perform an enormous number of training data for different shapes and locations.

Chapter 7

Plans for improvement

It would be necessary to make the operation of the neural network even more precise, for which it seems expedient to implement the following ideas:

- It would be important to eliminate the data loss obtained from smoothing the input data. To accomplish this, an idea is not to only use reflected waves but also the amplitude and frequency of the waves. This would result in having a higher dimensional data set when training the neural network.
- Multiple frequency waves should also be used, thus the various simulation results obtained from them will certainly contain more information from which the position of the reflected objects is likely to be more accurately estimated.
- If we get a more accurate procedure, we should try to deduce the shapes themselves; we could try not to only estimate the positions of the spheres, but their radii as well.
- We could try placing more spheres in the grid too.

Appendix: the created program codes

7.1 Simulation for obtaining the data set

The program specified here is written in Matlab, relies on the subroutines of the `mu-diff` simulation program package, and contains the principle of simulation, described in Chapter 5.

```
function y = Pr()

%First index of the iteration
l = 1;

%Size of the grid in one direction
h = 4;
h_lepes=0.05;

%The locations of the spheres
d0 = h-1;

%The maximum possible spheres
d1 = (d0+1)^4;
A = h*2/h_lepes+1;

%Radius of spheres
a = [0.4, 0.4];

%Vector containing the input signal
z = zeros(d1,A);
z_mod_matr=z;

%Vector containing the centres of spheres
q = zeros(d1,5);

%Wave number
k = 4*pi;

%Radiation angle
incident_angle = pi/2;

```

```
M_modes = FourierTruncation(a, k, 'Min', 1);
RESTART = [];
TOL = 10^(-10);
MAXIT = sum(2*M_modes+1);
beta_inc = pi;
%Iterating through the possible centres
for i = -d0:1:d0
    for j = -d0:1:d0
        for m = i:1:d0
            for n = j:1:d0
                if m==i & n == j
                else
                    O = [i,m; j,n];
                    N_scat = size(O,1);

                    Solution = NeumannSolver(O, a, k, '
                       PlaneWave', incident_angle);

                    SpPrecond = SpIntegralOperator(O, a,
                       M_modes, k, {'Lprec'});
                    UincPrecond = PlaneWavePrecond(O, a,
                       M_modes, k, beta_inc);

                    [rho_SpPrecond,FLAG_SpPrecond,
                       RELRES_SpPrecond,ITER_SpPrecond,
                       RESVEC_SpPrecond] = gmres(@(X)SpMatVec(X
                       ,M_modes,SpPrecond), UincPrecond,
                       RESTART, TOL, MAXIT, [], []);

                    [X,Y] = meshgrid([-h:h_lepes:h], [-h:
                       h_lepes:h]);
                    [U_tot, U] = NeumannNearField(Solution, X,
                        Y);
                    z(l,:) = abs(U(1,:));
                    q(l,1) = O(1,1);
                    q(l,2) = O(1,2);
                    q(l,3) = O(2,1);
                    q(l,4) = O(2,2);
                    l = l+1
                end
            end
        end
    end
end

%Writing the results in a txt file
dlmwrite('NearFieldData_no_rep_4.txt',z,'delimiter',' ','
   newline','pc')
```

```
71 dlmwrite('Characteristics_no_rep_4.txt',q,'delimiter',' ','
       newline','pc')
72
73 end
```

7.2 Transforming the data set

The program given in this section transforms the results from the previous simulation, in order to make it easier to train NN. A description of this can also be found in Chapter 5.

```
1  function g = fontos_adat_wave(M)
2
3  %M matrix: kth row of the reflected wave
4  sizeM = [600 inf];
5  fileID = fopen('NearFieldData.txt','r');
6  formatSpec = '%f';
7  M = fscanf(fileID, formatSpec, sizeM);
8  fclose(fileID);
9
10 si=size(M);
11 M_w_interp = [];
12 M_ampl_interp = [];
13
14 for j=1:si(1)
15 w=M(j,:);
16
17 %Saving w
18 w_orig = w;
19 onw=ones(length(w),1);
20 A1=spdiags([onw, -onw],0:1,length(w),length(w));
21
22 %Subtracting elements after each other
23 %Plus sign: decrease
24 %Minus sign: increase
25 w=sign(A1*w');
26
27 A2=spdiags([onw, onw],0:1,length(w),length(w));
28
29 %Numerical derivation
30 w=A2*w;
31
32 %Minimums, maxima
33 w=find(~w);
34
35 A1=A1([1:length(w)],[1:length(w)]);
36 A2=A2([1:length(w)],[1:length(w)]);
37
```

```
%Half wave lengths
w_w = A1*w;
w_w(end)=[];
w_w(end)=[];

%Centre of half wave lengths
w_mp = floor(A2*w/2);
w_mp(end)=[];
w_mp(end)=[];

%Extreme value of wave
w_orig(w);

%Amplitudes
w_ampl=abs(A1*w_orig(w)');
w_ampl(end)=[];
w_ampl(end)=[];

%Half wave length on an equally distributed grid point
w_w_interp=interp1(w_mp,w_w, [4:4:111]);

%Amplitude wave length on an equally distributed grid point
w_ampl_interp=interp1(w_mp,w_ampl, [4:4:111]);

M_w_interp = [M_w_interp; w_w_interp];
M_ampl_interp = [M_ampl_interp; w_ampl_interp];
end

%Writing data into txt files
dlmwrite('amplitude_scattered.txt',M_ampl_interp,'delimiter',' ','newline','pc')
dlmwrite('wlength_scattered.txt',M_w_interp,'delimiter',' ','newline','pc')

end
```

7.3 Python code with the neural network

There are multiple notebooks created for this research, however, only the main one is presented in this appendix.

Short description of the notebook:

This notebook was created and ran in google colab.

The following notebook imports three txt files: - interp_scattered_Noisy_NearFieldData: dataset of the wave intensities on - Characteristics_no_rep.txt: geometrical locations of the scattering objects

The data is preprocessed with the preprocessing.scale() function, which centres the data set to the mean and scales it with the variance. The data is then reduced to shorter vectors, so the NN could learn the different features more easily. Right after, a function is created to split the data set into training and testing data sets with a random integer used as a random_state in the splitting. Finally, the NN is created and called on the split data set. The parameter summary of the model can be seen in the end of the notebook, just as the plot of the validation loss.

```
1
2  ## Importing the necessary libraries
3
4  # Basic calculations and plotting
5  import numpy as np
6  import matplotlib.pyplot as plt
7
8  # Generating random numbers
9  import random
10
11 # Prerocessing the data, splitting the dataset, calculating
       the accuracy
12 from sklearn import preprocessing
13 import skimage.measure
14 from sklearn.model_selection import train_test_split
15 from sklearn.metrics import accuracy_score
16
17 # Creating the NN
18 from tensorflow import keras
19 from tensorflow.keras import layers
20 from tensorflow.keras.layers import Permute, Dense, Activation
       , LocallyConnected2D, Reshape, Flatten, Dropout, Input
21 from tensorflow.keras.models import Sequential, Model
22 import tensorflow.keras.optimizers
23
24
25 ## Loading and preprocessing the data
26
27 # Importing the dataset
28
29 x_all = np.genfromtxt("interp_scattered_Noisy_NearFieldData.
       txt", delimiter='')
30 y = np.genfromtxt("Characteristics_no_rep.txt", delimiter='')
31
32 # Setting the number of angles
33
34 number_of_angles = 16
35
36 # We can delete the last column of the imported dataset, cause
       we don't calculate with the wavelength just yet
37
```

```
38 y = np.delete(y, [4], axis = 1)
39
40 # We transform the y-coordinates, so the norm is one
41
42 y = y+3
43 y = y/6
44
45 # Some examples for the wave dataset
46 # Selecting some random indices
47
48 indices = random.sample(range(1, len(x_all)), 10)
49
50 # Creating a plot of the results
51
52 fig, axs = plt.subplots(2,5, figsize=(25, 8), facecolor='w',
       edgecolor='k')
53 fig.subplots_adjust(hspace = 0.4, wspace = 0.4)
54
55 axs = axs.ravel()
56
57 for i in range(0,len(indices)):
58
59     axs[i].plot(x_all[indices[i]], color = 'black', label = '
          Original data')
60     axs[i].set_title('The label: ' + str(indices[i]))
61     axs[i].set_xlabel('Dimensionless distance on the grid')
62     axs[i].set_ylabel('Dimensionless intensity of wave')
63     axs[i].grid()
64     axs[i].legend()
65
66 plt.show()
67
68 # Creating some lists to store the new datasets
69
70 dataset = []
71 reduced_dataset = []
72
73 # Separating the dataset into a list of lists
74 # Preprocessing the dataset with the preprocessing.scale()
       function
75
76 for i in range(0,number_of_angles):
77     dataset.append(preprocessing.scale(x_all[735*(i):735*(i+1)
          ], axis=0))
78
79 # Reducing the size of the dataset with the skimage.measure.
       block_reduce() function
80 # Reducing the size of vectors from len = 161 to len = 11
```

```
81 # For this the splitting size must equal to 15
82 # Using np.average to estimate between the points
83
84 splitting_size = 15
85
86 for i in range(0,len(dataset)):
87     reduced_dataset.append(skimage.measure.block_reduce(
       dataset[i], (1,splitting_size), np.average))
88
89
90 # Generating some random indices
91
92 indices1 = random.sample(range(1, len(dataset)), 10)
93 indices2 = random.sample(range(1, len(dataset[0])), 10)
94
95 # Comparing the reduced and the noisy dataset
96
97 fig, axs = plt.subplots(2,5, figsize=(25, 8), facecolor='w',
     edgecolor='k')
98 fig.subplots_adjust(hspace = 0.4, wspace = 0.4)
99
100 axs = axs.ravel()
101
102 for i in range(0,len(indices1)):
103
104     axs[i].plot(dataset[indices1[i]][indices2[i]], color = '
       black', label = 'Noisy data')
105     axs[i].plot(np.linspace(0,161,11), reduced_dataset[
       indices1[i]][indices2[i]], color = 'orange', label = '
       Reduced data')
106     axs[i].set_title('The label: ' + str(indices1[i]))
107     axs[i].set_xlabel('Dimensionless distance on the grid')
108     axs[i].set_ylabel('Dimensionless intensity of wave')
109     axs[i].grid()
110     axs[i].legend()
111
112 #plt.savefig('images.jpg', dpi=800)
113 plt.show()
114
115
116 ## Function for splitting the dataset
117
118 # Splitting the matrix to train and test dataset
119 # Creating some lists to store the data
120 # Saving the train and test datasets to lists
121
122 def dataset_split(reduced_dataset, rand_state):
123
```

```
124     combined_vector_train = []
125     combined_vector_test = []
126 
127     for i in range(0,len(reduced_dataset)):
128         x_train, x_test, y_train, y_test = train_test_split(
              reduced_dataset[i], y, test_size = 0.08,
              random_state = rand_state)
129 
130         combined_vector_train.append(x_train)
131         combined_vector_test.append(x_test)
132 
133     # Changing the dimensions of the matrices, so they are
        appropriate for the NN
134 
135     combined_vector_train = np.swapaxes(combined_vector_train
        ,0,1)
136     combined_vector_test = np.swapaxes(combined_vector_test
        ,0,1)
137 
138     matrix_of_results = [combined_vector_train,
        combined_vector_test, y_train, y_test]
139 
140     return(matrix_of_results)
141 
142 
143 ## Creating the model and launching the neural network
144 
145 # Setting the parameters of the NN
146 
147 runs = 1
148 BATCH = 32
149 EPOCH = 250
150 
151 
152 for i in range (0,runs):
153 
154     # Generating a random number and using it as the
        random_state of the train-test splitting
155     random_state = random.randint(0,100)
156     # Calling the function of the splitting
157     results = dataset_split(reduced_dataset, random_state)
158 
159     # Saving the results in arrays
160     combined_vector_train = results[0]
161     combined_vector_test = results[1]
162     y_train = results[2]
163     y_test = results[3]
164 
```

```
165     # Creating the model
166     model = Sequential()
167
168     # Adding the layers
169     model.add(Permute((2,1), input_shape=(16,11)))
170     model.add(Reshape((16,11,1)))
171     model.add(LocallyConnected2D(12,(1,4), activation='relu',
            use_bias=True))
172     model.add(Dropout(0.5))
173     model.add(Flatten())
174     model.add(Dense(50, activation='relu'))
175     model.add(Dense(40, activation='relu'))
176     model.add(Dense(8, activation='relu'))
177     model.add(Flatten())
178     model.add(Dense(4,activation='sigmoid'))
179
180     # Setting the optimizer and compiling the model
181     optimizer = keras.optimizers.Adam(learning_rate=0.0005,
            beta_1=0.9, beta_2=0.8, amsgrad=False)
182     model.compile(loss='mse', optimizer=optimizer)
183
184     # Fitting the model
185     print("Run",i+1,"/",runs)
186     history = model.fit(combined_vector_train, y_train,
            validation_data=(combined_vector_test, y_test),
            batch_size=BATCH, epochs=EPOCH, verbose = 0)
187
188     # Calculating the different parameters of the model
189     min_of_val_loss = np.min(history.history['val_loss'])
190     index_of_min = history.history['val_loss'].index(
            min_of_val_loss)
191     val_loss_min = '{:0.4f}'.format(min_of_val_loss)
192     val_loss_final = '{:0.4f}'.format(history.history['
            val_loss'][EPOCH-1])
193
194     # Plotting the validation- and training loss of the model
195     plt.plot(history.history['loss'], color = 'orange', label=
            'train loss')
196     plt.plot(history.history['val_loss'], color = 'black',
            label='validation loss')
197     plt.xlabel('epochs', fontsize=15)
198     plt.ylim(0, 0.006)
199     plt.ylabel('loss', fontsize=15)
200     plt.legend(fontsize=15)
201     plt.grid()
202     plt.show()
203
204     # Predicting the test values of the dataset
```

```
205     predicted_values = np.matrix.round(model.predict(
            combined_vector_test)*6-3)
206     y_test_pred_flat = np.ndarray.flatten(predicted_values)
207     y_test_flat = np.ndarray.flatten(y_test)
208 
209     # Calculating the accuracy of the model
210     acc = accuracy_score(y_test_flat*6-3, y_test_pred_flat)
211 
212     # Printing the results
213     print('The accuracy of the model: ', round(acc,4))
214     print("Validation loss =", '{:0.4f}'.format(history.
            history['val_loss'][EPOCH-1]))
215     print('Minimum of validation loss:', '{:0.4f}'.format(
            min_of_val_loss), 'at EPOCH', index_of_min)
216     print('Random state = ', random_state)
217     print()
218 
219 
220 # Checking the parameter number of the model
221 
222 model.summary()
223 
224     # Plotting one graph of the trainin and validation loss
225 
226     plt.figure(figsize=(7,5))
227 
228     plt.plot(history.history['loss'], color = 'orange', label=
            'train loss')
229     plt.plot(history.history['val_loss'], color = 'black',
            label='validation loss')
230 
231     plt.xlabel('epochs', fontsize=15)
232     plt.ylabel('loss', fontsize=15)
233     plt.legend(fontsize=15)
234     plt.grid()
235 
236     plt.savefig('valid_loss.jpg', dpi = 800)
237 
238     plt.show()
```

Bibliography

[1] L. Bottou, F. E. Curtis, and J. Nocedal. Optimization methods for large-scale machine learning. *SIAM Rev.*, 60(2):223–311, 2018.

[2] D. Colton, J. Coyle, and P. Monk. Recent developments in inverse acoustic scattering theory. *SIAM Rev.*, 42(3):369–414, 2000.

[3] D. Colton and R. Kress. Looking back on inverse scattering theory. *SIAM Rev.*, 60(4):779–807, 2018.

[4] K.-L. Du and M. N. S. Swamy. *Neural networks and statistical learning*. Springer, London, 2014.

[5] M. Feigin, D. Freedman, and W. B. Anthony. A deep learning framework for single-sided sound speed inversion in medical ultrasound. *IEEE Trans Biomed Eng.*, 67(4):1142–1151, 2020.

[6] P. Filippi, D. Habault, J.-P. Lefebvre, and A. Bergassoli. *Acoustics: Basic Physics, Theory, and Methods*. Numerical Mathematics and Scientific Computation. Academic Press, New York, 1999.

[7] D. Haffner and F. Izsák. Localization of scattering objects using neural networks. *Sensors*, 21(1), 2021.

[8] D. Kingma and J. Ba. Adam: A method for stochastic optimization. In *Proceedings of the 3rd International Conference on Learning Representations (ICLR 2015)*, 2015.

[9] Complex representation and the Helmholtz equation. http://swissenschaft.ch/tesla/content/T_Library/L_Theory/EM{}Field{}Standard/Complex{}Representation{}and{}Helmholtz{}Equation.pdf, accessed 25.12.2006.

[10] P. Monk. *Finite element methods for Maxwell's equations*. Numerical Mathematics and Scientific Computation. Oxford University Press, New York, 2003.

[11] H. Pujol, E. Bavu, and A. Garcia. Source localization in reverberant rooms using deep learning and microphone arrays. Proceedings of the 23rd International Congress on Acoustics, 2019 Aachen, Germany.

[12] S. Shalev-Shwartz and S. Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, 2014.

[13] M. Siddharth, L. Hao, and H. Jiabo. *Machine Learning for Subsurface Characterization*. Gulf Professional Publishing, Elsevier, Cambridge, MA, 2020.

[14] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

[15] B. Thierry, X. Antoine, C. Chniti, and H. Alzubaidi. μ-diff: an open-source Matlab toolbox for computing multiple scattering problems by disks. *Comput. Phys. Commun.*, 192:348–362, 2015.

[16] J. Vera-Diaz, D. Pizarro, and J. Macias-Guarasa. Towards end-to-end acoustic localization using deep learning: From audio signals to source position coordinates. *Sensors*, 18(10):3418, 2018.

[17] M. Wes. *Python for Data Analysis*. O'Reilly Media, Inc., 1 edition, 2012.

[18] D. Yarotsky. Error bounds for approximations with deep relu networks. *Neural Networks*, 94:103 – 114, 2017.