The Dirichlet to Neumann map and its applications

MSc Thesis
January 2023

Taki Eddine Djebbar
Applied Mathematics MSc Student

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

Eötvös Loránd University, Faculty of Sciences

STATEMENT OF THESIS SUBMISSION AND ORIGINALITY

I hereby confirm the submission of the Master Thesis Work on the Applied Mathematics Msc course with author and title:

Name of Student: *Taki Eddine Djebbar*
Code of Student: *A19NGE*
Title of Thesis: The Dirichlet to Neumann map and its applications
Supervisor: *Dr. Ferenc Izsák*
Department of Departments : Applied Analysis and Computational Mathematics
At Eötvös Loránd University, Faculty of Science, Mathematical Institute, Department of Applied Analysis and Computational Mathematics

In the consciousness of my full legal and disciplinary responsibility, I hereby claim that the submitted thesis work is my own original intellectual product, and the use of referenced literature is done according to the general rules of copyright.
I understand that in the case of thesis works the following acts are considered plagiarism:

- literal quotation without quotation marks and reference;
- citation of content without reference;
- presenting others' published thoughts as own thoughts.

Budapest, Dec 31, 2022

Taki Eddine Djebbar
student

Acknowledgements

The first thing to say is that this thesis is dedicated to my mother and my family, for their continued support and belief in me.

I would like to extend my sincere gratitude to my supervisor Dr.Ferenc Izsák, for his enormous support, help, patience, and availability. His guidance and advice helped me through all stages of writing my thesis. As a result, it has inspired me to keep moving forward in this line of work and has given me the strength and desire to succeed.

Contents

Chapter 1

Introduction

1.1 General introduction

1.1.1 Motivation: the electrical impedance tomography

The motivation of our study is an important real-life measurement procedure, the electrical impedance tomography. In this section, we introduce this method and the corresponding mathematical model.

In general, in many non-invasive measurement procedures in the medicine, we collect information at the surface of an organ. At the same time, our aim is to infer to the entire structure of this organ. In concrete terms, in such a procedure, electrodes are placed around the surface of the organ investigated. Then some currents are applied to these and the resulting potential is measured between the electrodes. This is repeated by generating different current impulses and getting potentials.

In real life situations, the result, which we are looking for, is the impedance (or even the permittivity) of the organ. The result has usually to be visualized, which contains information on the structure of the investigated organ.

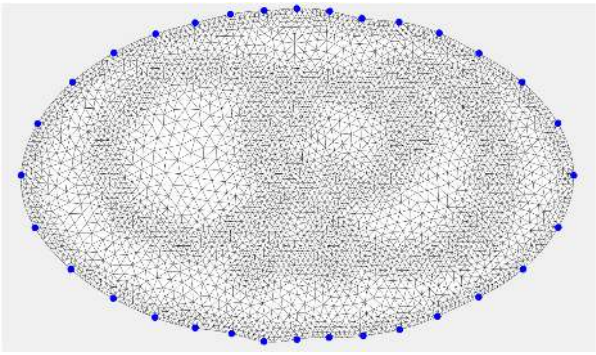


Figure 1.1: The domain Ω used for simulation of the EIT data corresponding to the small lateral pneumothorax, with 32 electrode centers plotted on the boundary.

1.1.2 The aim of the present work

The objective of the present thesis is the detailed study of the Dirichlet-to-Neumann problem from more points of view. Regarding the classical approach, we first have to collect tools of the classical analysis. Citing the literature, and pointing out the limitations of that approach, we propose something new. We intend to propose an alternative approach using the modern computing tool, the neural networks. In practice, we want to make use the most powerful packages of Python such as Keras, Tensorflow and Pytorch. For this, we also want to develop appropriate learning data and construct a feasible neural network, and tune its parameters for an efficient computation. In concrete terms, various setups (e.g., multiple layers) and parameters will be tested to have better performance with less time complexity and less parameters.
After performing numerical experiments, we want to evaluate the corresponding results.

1.1.3 The outline of the thesis

The structure of this thesis is as follows:

- Chapter 1 - This chapter covers all relevant physical background to the mathematical formulation of the problem.

- Chapter 2 - The scope of this chapter is to define the spaces on which the DtN map is defined. We define a fractional Sobolev space with $s = \frac{1}{2}$ and its dual space, which are the departure and arrival spaces of the DtN map respectively, as well as some of its important properties.

- Chapter 3 - This chapter presents the forward problem for finding the DtN map using directly the Dirichlet boundary condition, where we introduce the mathematical form using Green's function and then find the kernel for the DtN map. Then we discussed briefly the inverse problem which presented by Calderon problem, as well as the formula of the fundamental solution that will be used to train the machine learning algorithm.

- Chapter 4 -In this chapter, we introduce the basics of machine learning and some optimization techniques for training our model, then we test multiple models to determine which is the most accurate. As a final step, we keep three critical models to train our data, we denote approach 1 and approach 2 for the first two models where we give the Dirichlet data to the input of the NN and the Neumann data to the output. The third model we used different method, we try to predict the points inside the domain then use the directional derivative formula to approximate the DtN map.
  At the end of the chapter we make some comparisons between the three models.

- Chapter 5 - This chapter concludes the best model in terms of good estimation, fewer parameters and shorter time complexity.

1.2 Mathematical introduction

1.2.1 The equations in the model

In the basic problem, we investigate, u is the solution of the following conductivity equation

$$L_\gamma u = \operatorname{div}(\gamma \nabla u) = 0 \quad \text{in } \Omega.$$

If this is equipped with Dirichlet boundary condition, i.e. we prescribe electric potential on the boundary using the function f, we get the problem

$$\begin{cases} L_\gamma u = 0 & \text{in } \Omega \\ u|_{\partial\Omega} = f. \end{cases} \tag{1.1}$$

Using the Dirichlet boundary condition again, we also mention a similar problem called the Schrödinger equation. To find the conductivity, we will use the Schrödinger equation. Our goal is to prove that these two problems are related. The Schrödinger equation is given by

$$\begin{cases} L_\eta u = 0 \\ u|_{\partial\Omega} = f, \end{cases} \tag{1.2}$$

where the differential operator is defined with

$$L_\eta u := \Delta u - \eta u,$$

where η is constant and f describes again the potential on the boundary. Now we want to investigate the relationship between these two problems, Λ_γ and Λ_η are correlated in the following way

$$\omega = \gamma^{-\frac{1}{2}} u$$

if u is the solution of the conductivity equation then ω is the solution of the Schrödinger equation with $\eta = \frac{\Delta\sqrt{\gamma}}{\sqrt{\gamma}}$, the next lemma will show how the two problems close (related) to each other.

Lemma 1. *Let $\gamma \in \mathcal{C}\left(\bar{\Omega}\right)$ be a positive conductivity function then we have*

$$\gamma^{-\frac{1}{2}} L_\gamma \left(\gamma^{-\frac{1}{2}}\right) = \Delta - \eta,$$

with

$$\eta = \frac{\Delta\left(\gamma^{\frac{1}{2}}\right)}{\gamma^{\frac{1}{2}}}. \tag{1.3}$$

Proof since

$$L_\gamma u = \gamma \Delta u + \nabla\gamma \cdot \nabla u$$

we obtain

$$\gamma^{-\frac{1}{2}} L_\gamma u = \gamma^{\frac{1}{2}} \Delta u + \frac{\nabla\gamma \cdot \nabla u}{\gamma^{\frac{1}{2}}}.$$

taking $w = \gamma^{\frac{1}{2}} u$ use then 1.3, rewriting the differential operator three times we get

$$\begin{aligned}
\Delta w - \eta w &= \Delta\left(\gamma^{\frac{1}{2}} u\right) - \left(\Delta\gamma^{\frac{1}{2}}\right) u \\
&= \nabla \cdot \left(\nabla\left(\gamma^{\frac{1}{2}} u\right)\right) - \left(\Delta\gamma^{\frac{1}{2}}\right) u \\
&= \nabla \cdot \left(\left(\nabla\gamma^{\frac{1}{2}} u\right) + \gamma^{\frac{1}{2}}(\nabla u)\right) - \left(\Delta\gamma^{\frac{1}{2}}\right) u \\
&= \left(\Delta\gamma^{\frac{1}{2}}\right) u + 2\nabla\gamma^{\frac{1}{2}} \cdot \nabla u + \gamma^{\frac{1}{2}}\Delta u - \left(\Delta\gamma^{\frac{1}{2}}\right) u.
\end{aligned}$$

using the chain rule we also have

$$\nabla\gamma^{\frac{1}{2}} = \nabla\gamma \cdot \frac{1}{2} \cdot \gamma^{-\frac{1}{2}},$$

such that we get

$$\nabla w - \eta w = \gamma^{\frac{1}{2}}\Delta u + \frac{\nabla\gamma \cdot \nabla u}{\gamma^{\frac{1}{2}}} = \gamma^{\frac{1}{2}} L_\gamma u$$

see [3] on the page [3, p. 17].

1.2.2 The Dirichlet to Neumann map

The DtN map associate the current density on the boundary $\partial\Omega$ to the given (or prescribed) voltage on the boundary. In practice, even the inverse of this mapping can be measured at some certain points by applying some currents on the boundary and measuring the voltages as a response.

Definition 1.2.1. *The Dirichlet to Neumann map is given by*

$$\Lambda_\gamma : H^{\frac{1}{2}}(\partial\Omega) \to H^{-\frac{1}{2}}(\partial\Omega) \quad \textit{with} \quad \Lambda_\gamma(f) = \gamma\partial_{\boldsymbol{\nu}} u.$$

Here $\boldsymbol{\nu}$ denotes the unit outward normal vector on $\partial\Omega$. In a similar way, Dirichlet to Neumann map Λ_η associated to the Schrödinger equation (1.2) is defined as

$$\wedge_\eta(f) = \partial_{\boldsymbol{\nu}} u.$$

Example 1.2.1. *This example was derived from [8]*
- Let $u : \mathbb{R}^n \to \mathbb{R}$.
- Extend it harmonically to $\mathbb{R}^{n+1}_+$

$$-\Delta\mathcal{U} = 0, \textit{ in } \mathbb{R}^{n+1}_+, \mathcal{U}(\cdot, 0) = u.$$

- The Dirichlet to Neumann map is

$$\mathrm{DtN} : u \mapsto -\partial_y \mathcal{U}(\cdot, 0).$$

the DtN has the following properties:

- $\mathrm{DtN}^2 = -\Delta$: Indeed, since $-\Delta_{x',y}\mathcal{U} = -\Delta_{x'}\mathcal{U} - \partial_y^2\mathcal{U} = 0$,

$$\mathrm{DtN}\, u = -\partial_y \mathcal{U}(\cdot, 0)$$

$$\mathrm{DtN}^2\, u = -\partial_y \left(-\partial_y \mathcal{U}(\cdot, 0)\right) = -\Delta_{x'}\mathcal{U}(\cdot, 0) = -\Delta_{x'} u.$$

- DtN is positive: Since $\mathcal{U}$ is harmonic

$$0 = -\int_{\mathbb{R}^{n+1}_+} \Delta\mathcal{U}\mathcal{U}\mathrm{d}x\ \mathrm{d}y = \int_{\mathbb{R}^{n+1}_+} |\nabla\mathcal{U}|^2\ \mathrm{d}x\ \mathrm{d}y + \int_{\mathbb{R}^n} \partial_y\mathcal{U}\mathcal{U}\mathrm{d}x.$$

then

$$\int_{\mathbb{R}^{n+1}_+} |\nabla\mathcal{U}|^2\ \mathrm{d}x\ \mathrm{d}y = -\int_{\mathbb{R}^n} \partial_y\mathcal{U}\mathcal{U}\mathrm{d}x.$$

we know that

$$\int_{\mathbb{R}^n} u\,\mathrm{DtN}\, u\ \mathrm{d}x = -\int_{\mathbb{R}^n} \partial_y\mathcal{U}\mathcal{U}\mathrm{d}x > 0$$

Thus, we define

$$\mathrm{DtN} = (-\Delta_x)^{\frac{1}{2}}, \quad (-\Delta_x)^{\frac{1}{2}}\, u = \partial_\nu \mathcal{U}.$$

Chapter 2

Functional analysis for the DtN map

In the entire section, $\Omega \subset \mathbb{R}^n$ denotes a bounded open set with piecewise Lipschitz boundary, also, ∂_j denotes the generalized derivatives with respect to the j th variable.

2.1 Hilbert space

Definition 2.1.1. *Let H be a normed vector space on the field $K = (\mathbb{R}$ or $\mathbb{C})$. We say that a map $\langle \cdot, \cdot \rangle$ is an inner product in H if it is bilinear (sesquilinear), symmetric, and positive definite.*
this means that

$$\langle \cdot, \cdot \rangle : H \times H \to K$$
$$(x, y) \to \langle x, y \rangle$$

satisfies the following: Linearity: for $x, y, z \in H$ and $a, b \in K$

$$\langle ax + by, z \rangle = a\langle x, z \rangle + b\langle y, z \rangle.$$

Conjugate symmetry: for $x, y \in H$

$$\langle x, y \rangle = \overline{\langle y, x \rangle}.$$

Positive definiteness: for $x \in H$

$$\langle x, x \rangle \geq 0 \text{ and we have also } \langle x, x \rangle = 0 \implies x = 0.$$

From now on we will use the inner product in L^2 which is given by

$$(u, v) = \int_\Omega u\bar{v},$$

for all u, v in $L^2(\Omega)$.

Definition 2.1.2 (Hilbert space). *We call H a Hilbert space if it is formed by an inner product (in this case it is called pre-Hilbert space), and it is complete.*

2.2 Sobolev Spaces

Definition 2.2.1. ***(Sobolev space)*** *The Sobolev space $H^1(\Omega)$ is the following real Hilbert space:*

$$H^1(\Omega) := \left\{ u \in L^2(\Omega) : \quad \partial_i u \in L^2(\Omega) \right\}.$$

It is equipped with the following inner product and induced norm, respectively:

$$\langle u, v \rangle_{H^1} := \int_\Omega (\nabla u \cdot \nabla v + uv), \quad \|u\|^2_{H^1} := \int_\Omega \left(|\nabla u|^2 + u^2 \right).$$

Definition 2.2.2. *We denote by $H^1_0(\Omega)$ the closure of $\mathcal{D}(\Omega) = C^\infty_0(\Omega)$ in $H^1(\Omega)$. By extension, we note $H^m_0(\Omega)$ the closure of $\mathcal{D}(\Omega)$ in $H^m(\Omega)$.*

Theorem 2.2.1. *Consider Ω an open set of class C^1. Then there is the continuous linear operator, called trace operator, which is denoted by γ_0 and maps $H^1(\Omega)$ into $L^2(\partial\Omega)$, that is consistent with continuous function restriction operators $\left(\boldsymbol{tr}\ u = u|_{\partial\Omega}\right.$ for all $\left. u \in C(\overline{\Omega}) \cap H^1(\Omega)\right)$. Its kernel is $\operatorname{Ker}(\gamma_0) = H^1_0(\Omega)$.*

Theorem 2.2.2. ***(Trace)*** *The trace operator refers to a continuous linear map that can be represented in the following way:*

$$\gamma_0 : H^1(\Omega) \to H^{1/2}(\partial\Omega),$$

By extending the restriction mapping of continuous functions, a trace mapping is created. This mapping is surjective and its kernel is $\operatorname{Ker}(\gamma_0) = H^1_0\left(\mathbb{R}^d_+\right)$.

Definition 2.2.3. *Let us consider the $C^m(\Omega), m \in \mathbb{N}^*$ and γ_0 the trace operator in the previous theorem. We define the space:*

$$H^{m-1/2}(\partial\Omega) = \gamma_0\left(H^m(\Omega)\right),$$

endowed with the norm:

$$\|u\|_{H^{m-1/2}(\partial\Omega)} = \inf_{v \in \gamma_O^{-1}(\{u\})} \|v\|_{H^m(\Omega)}.$$

Definition 2.2.4. *The Sobolev space $H^{1/2}$ is just a special case for the previous definition 2.2.3 when $m = 1$. See [18],[14] for more detail about this space. The $H^{-1/2}$ is defined as the dual space of $H^{1/2}$ [13], with the norm*

$$\|u\|_{H^{-s}(\partial\Omega)} \sim \sup_{0 \neq v \in H^s(\partial\Omega)} \frac{|\langle u, v \rangle_{\partial\Omega}|}{\|v\|_{H^s(\partial\Omega)}} = \sup_{0 \neq v \in H^s(\partial\Omega)} \frac{|(u, v)_{\partial\Omega}|}{\|v\|_{H^s(\partial\Omega)}} \quad \text{for } |s| \leq 1,$$

where

$$\langle u, v \rangle_{\partial\Omega} = \int_{\partial\Omega} u(x) v(x)\, d\sigma(x) \quad \text{and} \quad (u, v)_{\partial\Omega} = \int_{\partial\Omega} \overline{u(x)} v(x)\, d\sigma(x),$$

for $s = \frac{1}{2}$.

2.3 Weak derivatives

In this section, we will define the normal derivative for certain functions in H. First, we recall the weak derivative of the Laplace operator. We also need the following simple version of Gauss's theorem.

Theorem 2.3.1. *Assume that $\Omega \subseteq \mathbb{R}^n$ is open and bounded and has C^1-boundary. There exists a unique Borel measure σ on $\partial\Omega$, the surface measure on $\partial\Omega$, such that*

$$\int_\Omega \partial_j u(x)\,\mathrm{d}x = \int_{\partial\Omega} u(z)\nu_j(z)\,\mathrm{d}\sigma(z),$$

for each $j \in \{1,\dots,n\}$ and all $u \in C^1(\bar{\Omega})$. Here $\nu \in C\left(\partial\Omega;\mathbb{R}^n\right)$ is the outer normal with coordinates $\nu(z) = (\nu_1(z),\dots,\nu_n(z))$.
For more details and applications about this theorem refer to [5].

Corollary 2.3.1.1 (Green's formulas). *Let $u \in C^2(\bar{\Omega})$. Then*

$$\int_\Omega (\Delta u)v\ \mathrm{d}x + \int_\Omega \nabla u\cdot\nabla v\ \mathrm{d}x = \int_{\partial\Omega} (\partial_\nu u)\,v\ \mathrm{d}\sigma \quad \left(v \in C^1(\bar{\Omega})\right). \tag{2.1}$$

$$\int_\Omega (v\Delta u - u\Delta v)\mathrm{d}x = \int_{\partial\Omega} (v\partial_\nu u - u\partial_\nu v)\,\mathrm{d}\sigma \quad \left(v \in C^2(\bar{\Omega})\right). \tag{2.2}$$

Proof. Using the above theorem 2.3.1, we will get

$$\int_\Omega \partial_j u\partial_j v = -\int_\Omega \left(\partial_j^2 u\right)v + \int_\Omega \partial_j\left((\partial_j u)\,v\right) = -\int_\Omega \left(\partial_j^2 u\right)v + \int_{\partial\Omega} (\partial_j u)\,v\nu_j\ \mathrm{d}\sigma.$$

We sum up over j to get 2.1. Flipping u and v in 2.1 and taking their difference gives simply 2.2. □

Let $\Omega \subseteq \mathbb{R}^n$ be open. Let $u, f \in L_2(\Omega)$. Then $\Delta u = f$ if

$$\int_\Omega u\Delta\varphi = \int_\Omega f\varphi \quad (\varphi \in C_c^\infty(\Omega)).$$

For $u \in L_2(\Omega)$ we say that $\Delta u \in L_2(\Omega)$ if there exists $f \in L_2(\Omega)$ such that $\Delta u = f$. Using Green's formula 2.3.1.1, we can define a normal derivative in a weak sense.

Let Ω be bounded, with C^1-boundary (the solution u is in C^1 on the boundary), and let $u \in H^1(\Omega)$ such that $\Delta u \in L_2(\Omega)$. We say that $\partial_\nu u \in L_2(\partial\Omega)$ if there exists $h \in L_2(\partial\Omega)$ such that

$$\int_\Omega (\Delta u)v + \int_\Omega \nabla u\cdot\nabla v = \int_{\partial\Omega} hv \quad \left(v \in H^1(\Omega)\right).$$

In that case $\partial_\nu u := h$. The trace sign was omitted from the integral over $\partial\Omega$. It is always the Lebesgue measure that governs the integral over Ω, and the surface measure governs the integral over $\partial\Omega$.

2.4 The adjoint operator of the DtN map

The DtN was applying to many problems and its properties can be changed from problem to another, in this part here we prove that DtN map is self-adjoint for the Laplacian and for the Schrödinger operator $\Delta + \eta$ as well (this section is more detailed in [5]).

Definition 2.4.1. *A function u is called harmonic if it is a twice continuously differentiable function and its Laplacian is equal to zero. The Laplacian operator is a second-order differential operator in the n-dimensional Euclidean space and it is given by*

$$\Delta f = \sum_{i=1}^{n} \frac{\partial^2 f}{\partial x_i^2}.$$

Definition 2.4.2. *Let $a : V \times V \to \mathbb{C}$ be a bounded bilinear form and let $j : V \to H$ linear with a dense range. We say a is a quasi-coercive with respect to j, if there exist $\omega \in \mathbb{R}, \alpha > 0$ such that*

$$\operatorname{Re} a(v) + \omega \|j(v)\|_H^2 \geqslant \alpha \|v\|_V^2 \quad (v \in V).$$

2.4.1 The Dirichlet-to-Neumann operator for the Laplacian

We consider classical Dirichlet form

$$a : H^1(\Omega) \times H^1(\Omega) \to \mathbb{R}$$

$$a(u, v) = \int_\Omega \nabla u \cdot \overline{\nabla v} \quad \left(u, v \in H^1(\Omega)\right). \tag{2.3}$$

Definition 2.4.3. *Let us consider j is the trace operator from $H^1(\Omega)$ to $L^2(\partial\Omega)$, by taking a is quasi-coercive, as a result we will get an associated operator in $L^2(\partial\Omega)$ is therefore self-adjoint. This operator turns out to be Dirichlet-to-Neumann operator Λ which maps $f \in L_2(\partial\Omega)$ to $\partial_\nu u \in L_2(\partial\Omega)$. Where $u \in H^1(\Omega)$ is the harmonic function with $u|_{\partial\Omega} = f$.*

Theorem 2.4.1. ***(Rellich-Kondrachov)*** *Let $\Omega \subseteq \mathbb{R}^n$ be open, bounded, with j Lipshitz boundary. Then the embedding $j : H^1(\Omega) \to L_2(\Omega)$ is compact.*

Theorem 2.4.2. *Take j to be the trace operator, and a to be the classical Dirichlet form. Then the operator Λ in $L_2(\partial\Omega)$ associated with (a, j) is described by*

$$\Lambda = \left\{(f, h) \in L_2(\partial\Omega) \times L_2(\partial\Omega); \exists u \in H^1(\Omega) : \Delta u = 0, u|_{\partial\Omega} = f, \partial_\nu u = h\right\}. \tag{2.4}$$

The operator associated Λ is self-adjoint and positive and has compact resolvent. Λ is called the Dirichlet to Neumann operator with respect to Δ.

Proof. This theorem will be proved by considering it as an equivalent of two statements.
We will first prove that for all (g, h) associated with (a, j) and for a set Λ implies that the operator Λ is a Dirichlet to Neumann operator, and it is defined

in (2.4).
In the second implication, we assume that we have the Dirichlet to Neumann map given in (2.4), then we try to conclude that the DTN operator was associated with (a, j) and we will just find the form (a, j) and the trace operator j.

1. ($\Longrightarrow$) Let $(g, h) \in \Lambda$ is associated with (a, j) the classical Dirichlet form. Then there exists $u \in H^1(\Omega)$ such that $u|_{\partial\Omega} = g$ (due to j is a trace operator) and we have the following formula by applying the divergence theorem
$$\int_\Omega \nabla u \cdot \overline{\nabla v} = a(u, v) = \int_{\partial\Omega} h\bar{v},$$
As this formula is equivalent to $-\Delta u = 0$ for $v \in \mathrm{C}_c(\Omega)$ this is valid also for $v \in H^{-1}(\Omega)$. Then we will add $\Delta u = 0$ to this form and using Green's formula in (2.3.1.1), we will have
$$\int_\Omega (\Delta u)\bar{v} + \int_\Omega \nabla u \cdot \overline{\nabla v} = \int_{\partial\Omega} h\bar{v}.$$
Finally we use the first equation from corollary 2.3.1.1. we get then $\partial_{\boldsymbol{\nu}} u = h$ so the first implication is true.

2. ($\Longleftarrow$) Let $u \in H^{-1}(\Omega)$ assuming that $u|_{\partial\Omega} = g$ and $h = \partial_{\boldsymbol{\nu}} u$. Then we have the following Green's formula
$$\int_\Omega \nabla u \cdot \overline{\nabla v} = \int_\Omega \nabla u \cdot \overline{\nabla v} + \int_\Omega (\Delta u)\bar{v} = \int_{\partial\Omega} h\bar{v} \quad \left(v \in H^1(\Omega)\right).$$
Here we can take $j(v) = \overline{v}$ (which is the trace operator) such that $a(u, v) = (h, j(v))_{L_2(\Omega)}$, . Therefore $(g, h) \in \Lambda$ by definition. □

Remark 1. *Using the symmetry of the classical Dirichlet form implies that* Λ *is self-adjoint, and dragging out the trace operator on the defined operator* j *in theorem 2.4.1 then we get that the tr operator is compact, therefore,* Λ *has a compact resolvent.*

2.4.2 The Dirichlet -to-Neumann operator for the Schrödinger operator

We again assume that $\Omega \subseteq \mathbb{R}^n$ has C^1-boundary and consider a function $\eta \in L_\infty(\Omega)$, we proceed similarly as we did in section 2.4.1. In this section, we will show that if zero is not an eigenvalue of $\Delta + \eta$ then the Schrödinger operator $\Delta + \eta$ is self-adjoint and it is bounded furthermore, it has a compact resolvent. Let us define the form $a : H^1(\Omega) \times H^1(\Omega) \to \mathbb{C}$ with
$$a(u, v) = \int_\Omega \nabla u \cdot \overline{\nabla v} - \int_\Omega \eta u\bar{v} \quad \left(u, v \in H^1(\Omega)\right).$$

Theorem 2.4.3. *Let V, H be Hilbert spaces, and let $a : V \times V \to \mathbb{K}$ be a continuous sesquilinear form. Let $j \in \mathcal{L}(V, H)$ has dense range. We assume that $u \in \ker(j)$. Then*

$$a(u, v) = 0 \text{ for all } v \in \ker(j) \text{ implies } u = 0. \tag{2.5}$$

Let

$$A := \{(x, y) \in H \times H; \exists u \in V : j(u) = x, a(u, v) = \langle y, j(v)\rangle (v \in V)\}$$

Definition 2.4.4. *Let H be a Hilbert space over $\mathbb{K}$. An operator A in H is called accretive (or definite positive) if*

$$\mathrm{Re}(Ax \mid x) \geqslant 0 \quad (x \in \mathrm{dom}(A)).$$

An accretive operator A satisfying $ran(I + A) = H$ is called m-accretive.

Theorem 2.4.4. *Let $a : V \times V \to \mathbb{K}$ be a bounded form, and let $j \in \mathcal{L}(V, H)$ have dense range. Consider the case where (2.5) holds and a is compactly elliptic.*

Let A be the operator associated with (a, j). Then A is m-accretive. If a is symmetric, then A is self-adjoint and bounded from below. If j is compact, then A has compact resolvent.

For the proof, see[5]. Theorem 8.11.

Theorem 2.4.5. *If the Schrödinger operator does not have an eigenvalue equal to zero then the operator associated to*

$$\Lambda_\eta := \{(f, h) \in L_2(\partial\Omega) \times L_2(\partial\Omega); \exists u \in H^1(\Omega) : \Delta u + \eta u = 0, u|_{\partial\Omega} = f, \partial_{\boldsymbol{\nu}} u = h\},$$

has a compact resolvent, it is self-adjoint and positive. Furthermore Λ_η is bounded from below.

Proof. We will use the same idea as in theorem 2.4.2. First, we will prove that the statement in (2.5) is satisfied.
Let $u \in \ker(j)$ such that

$$a(u, v) = \int_\Omega \nabla u \cdot \overline{\nabla v} - \int_\Omega mu\bar{v} = 0 \text{for all} v \in \ker(j) = H_0^1(\Omega).$$

By assuming that zero is not an eigenvalue for $(\Delta + \eta)$ $\Delta m + \eta m$=0. Then $u = 0$.
For $(g, h) \in L_2(\Omega)$ we have $(g, h) \in \Lambda_{\boldsymbol{\nu}}$ if and only if $u \in H^1$ with $u|_{\partial\Omega} = g$ and

$$\int_\Omega \nabla u \cdot \overline{\nabla v} - \int_\Omega \eta u\bar{v} = \int_{\partial\Omega} h\bar{v}. \tag{2.6}$$

Using the fact that this formula is equivalent to $-\eta u - \Delta u = 0$ for $v \in \mathrm{C}_c(\Omega)$. Then putting $\eta u = -\Delta u$ in (2.6), and using Green's formula we get at the end $\partial_{\boldsymbol{\nu}} u = h$, thus $(g, h) \in \Lambda_\eta$.

Now we are going to prove the other implication of the theorem as we did in theorem 2.4.2, if $(g,h) \in \Lambda_\eta$ then we have $(\Delta + \eta) = 0$ and $u|_{\partial\Omega} = g$ and $\partial_{\boldsymbol{\nu}} u = h$ using Green's formula

$$\int_{\partial\Omega} h\bar{v} = \int_\Omega \nabla u \cdot \overline{\nabla v} + \int_\Omega (\Delta u)\bar{v} = \int_\Omega \nabla u \cdot \overline{\nabla v} - \int_\Omega \eta u \bar{v} = a(u,v) \quad \left(v \in H^1(\Omega)\right).$$

In this way $(g,h) \in \Lambda_\eta$. Using theorem 2.4.4 we conclude that Λ_η is a positive and Selfa-adjoint operator moreover it is bounded from below.

Chapter 3

The analytical solution for the DtN map

The DtN operator has a kernel distribution which we will denote by K and is defined as adistribution on $K : \partial\Omega \times \partial\Omega \to \mathbb{R}$. Using this, the DtN operator can be given by the formula

$$DtN(f)(x) = \int_{\partial\Omega} K(x,y)f(y)\,\mathrm{d}y.$$

The next challenge for us is how to get the kernel. Also, it is not clear whether we should take the forward problem for the DtN map to find $K(x,y)$ starting from a given $\eta(x)$ presents the charge in the Schroüdinger equation).
However the main goal of the EIT problem is to find the conductivity distribution $\gamma(x)$. The Schrödinger equation at zero energy held the same information as it was illustrated in 1.2.1, considering the Schrödinger equation, then it is enough to find the kernel for solving DtN problem and find $\Lambda(f)$.

The Green's function can be used for solving the inhomogeneous linear ODEs and PDEs with initial or boundary conditions. Green's function is a calculated operator acts with the right hand side of a PDE equation to find its solution u. To make it more clearly, let $\mathcal{L}$ be a linear differential operator.

Definition 3.0.1. *The function $G : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ is called a* Green's function *if it is a solution of*

$$\mathcal{L}G(x,y) = \delta(x-y) \qquad x \in \Omega, \tag{3.1}$$

where δ denotes the Dirac distribution and for a fixed $y \in \Omega$ (for more details refer to [19]).

Remark: The Dirac distribution δ is often called delta function. Whenever, it is not a function in the classical sense, the following formal properties are frequently used:

$$\int_{-\infty}^{\infty} \delta(x)\mathrm{d}x = 1. \tag{3.2}$$

$$\int_{-\infty}^{\infty} f(x)\delta\left(x-y\right)\mathrm{d}x = f\left(\mathrm{y}\right). \tag{3.3}$$

For more details about its properties refer to [16].

Now we need to know how can we implement the Green's function in our problem. We know that the DtN map is linear also in the Schrödinger problem. Then using this fact and the definition of the Dirac distribution, we will get the solution u based on the Green's function. Let us consider the following PDE problem:

$$\mathcal{L}u(x) = f(x) \tag{3.4}$$

By multiplying both sides in the equation (3.1) by $f(y)$ and integrate them we will get

$$\int \mathcal{L}G(x,y)f(y)ds = \int \delta(x-y)f(y)dy.$$

Using the property in (3.3), we obtain $f(y)$ in the RHS and for the reason that $\mathcal{L}$ is linear corresponds to x and not depends on y we can put $\mathcal{L}$ out of the integral in the left side, this can be reformulated to

$$\mathcal{L}u(x) = \mathcal{L}\int G(x,y)f(y)dy.$$

With this, the solution u of (3.4) can be given as

$$u(x) = \int G(x,y)f(y)dy. \tag{3.5}$$

Overall we use the following statement.

Theorem 3.0.1 (Divergence theorem). *If S is the boundary of a region E in $\mathbb{R}^d$ and $\vec{F} : \mathbb{R}^d \to \mathbb{R}^d$ is a differentiable vector field, then*

$$\iiint_B \operatorname{div}(\vec{F})\, \mathrm{d}S = \iint_S \vec{F} \cdot \eta \, \mathrm{d}.$$

Using the above tools, we present an approach to construct the DtN analytically.
Let's consider the Schrödinger equation and the operator $\mathcal{L} = \Delta + \eta$. The operator which would be constructed by Green's function is $\mathcal{G} = \mathcal{L}^{-1}$ which by (3.5) can be written as

$$\mathcal{G}f(x) = u(x) = \int_\Omega G(x,y)f(y)\mathrm{d}y.$$

Where G is the Green's function of $\mathcal{L}$ applied to the Dirichlet boundary condition.

To have a good intuition about the solution we first observe that

$$\int_{\partial\Omega} \frac{\partial u}{\partial n(y)}(y)G(x,y)\mathrm{d}S(y) = 0, \tag{3.6}$$

for $x \in \partial\Omega$ since the Green's function G is zero on the boundary.
Applying the divergence theorem to the vector field $\overrightarrow{F}(y) = (\nabla u(y) \cdot G(x,y))$ we get

$$0 = \int_\Omega \operatorname{div}_y \left(\nabla_y u(y) \cdot G(x,y)\right) \mathrm{d}y.$$

Expanding the divergence operator we get

$$0 = \int_\Omega \left(\Delta_y u(x,y) \cdot G(x,y) + \nabla_y G(x,y) \nabla_y u(x,y)\right) dy.$$

Now we apply the same theorem but in this time, the vector field is $\overrightarrow{F} = (\nabla_y G \cdot u(y))$, with the Dirichlet boundary conditions $u(x) = f(x)$ for $x \in \partial\Omega$. One application of the divergence theorem we get

$$\int_{\partial\Omega} \frac{\partial G}{\partial n(y)}(x,y) f(y) \mathrm{d}S(y) = \int_\Omega \operatorname{div}_y \left(\nabla_y G(x,y) \cdot u(y)\right) \mathrm{d}y. \tag{3.7}$$

By expanding the divergence we obtain

$$\int_{\partial\Omega} \frac{\partial G}{\partial n(y)}(x,y) f(y) \mathrm{d}S(y) = \int_\Omega \left(\Delta_y G(x,y) \cdot u(y) + \nabla_y G(x,y) \nabla_y u(y)\right) \mathrm{d}y$$

Using equality in (3.6) by substitute the second term in the integral

$$\int_{\partial\Omega} \frac{\partial G}{\partial n(y)}(x,y) f(y) \mathrm{d}S(y) = \int_\Omega \left(\Delta_y G(x,y) \cdot u(y) - \Delta_y u(y) \cdot G(x,y)\right) \mathrm{d}y$$

subtract a term in the integral then we are getting,

$$\begin{aligned} &\int_{\partial\Omega} \frac{\partial G}{\partial n(y)}(x,y) f(y) \mathrm{d}S(y) \\ &= \int_\Omega \left(-\left(-\Delta_y + \eta(y)\right) G(x,y) \cdot u(y) + \left(-\Delta_y + \eta(y)\right) u(y) \cdot G(x,y)\right) \mathrm{d}y \end{aligned}.$$

Since G is the Green function of $\mathcal{L} = -\Delta + \eta$. Using the property in (3.2) and $\mathcal{L}u = 0$ then

$$\int_{\partial\Omega} \frac{\partial G}{\partial n(y)}(x,y) f(y) \mathrm{d}S(y) = -u(x). \tag{3.8}$$

By taking the derivative with respect to the outward normal for $x \in \partial\Omega$ in (3.7) and (3.8), we get the formula

$$\frac{\partial u}{\partial n}(x) = -\int_{\partial\Omega} \frac{\partial^2 G}{\partial n(x) n(y)}(x,y) f(y) \mathrm{d}S(y), \quad x \in \partial\Omega,$$

since the Λ_η is a linear operator, the DtN map will be given by the formula

$$\Lambda f(x) = \frac{\partial u}{\partial n(x)} = \int_{\partial\Omega} K(x,y) f(y) \mathrm{d}y,$$

where

$$K(x,y) = -\frac{\partial^2 G}{\partial n(x) n(y)}(x,y) \qquad x, y \in \partial\Omega.$$

As denoted earlier, the function K displays the kernel function.

3.1 Fundamental solutions

This section is based on the results which were given by the lecture notes in [20]. The Dirichlet to Neumann operator can be found by applying Green's function to a PDE problem with Dirichlet boundary conditions, but the challenge is to find $G(\mathbf{x}, \mathbf{y})$ in the general case $\mathbb{R}^n$. It is not easy to get $G(\mathbf{x}, \mathbf{y})$ in $\mathbb{R}^n$, so we will take special domains in $\mathbb{R}^2$, using the same idea and extend it to a domain in $\mathbb{R}^n$ for $n \geq 1$.

Now let us consider the Laplace equation

$$\Delta u = 0. \tag{3.9}$$

Since the Laplacian is symmetric, we are looking for the radial solution to reduce our problem from PDE to an ODE which is easier to handle.
The fundamental solution for 3.9 in $\mathbb{R}^2$ will be given by:

$$\Delta \Gamma(\mathbf{x}) = \delta(\mathbf{x}),$$

where $\Gamma(\mathbf{x})$ is equal to $G(\mathbf{x}, \mathbf{y})$ for fixed $\mathbf{y}$ at the origin. For the derivation, we will extend Section 2.2.1 in [10].
Due to the rotational invariance of the Laplacian, $\Gamma(\mathbf{x})$ will be radially symmetric, that is:

$$\Gamma(\mathbf{x}) = \Gamma(r), \quad r = \sqrt{x^2 + y^2}.$$

Theorem 3.1.1. *Let $x = r\cos\theta$ and $y = r\sin\theta$, where $r \geq 0$ and $\theta \in [0, 2\pi]$. Then the Laplace operator Δ can be transformed to polar coordinates by the following formula:*

$$\Delta u(x, y) = \frac{1}{r}\frac{\partial u(r,\theta)}{\partial r} + \frac{1}{r^2}\frac{\partial^2 u(r,\theta)}{\partial \theta^2} + \frac{\partial^2 u(r,\theta)}{\partial r^2}.$$

Proof. Using the chain rule we get

$$\begin{aligned}
\frac{\partial u}{\partial r} &= \frac{\partial u}{\partial x}\frac{\partial x}{\partial r} + \frac{\partial u}{\partial y}\frac{\partial y}{\partial r} \\
&= \frac{\partial u}{\partial x}\cos\theta + \frac{\partial u}{\partial y}\sin\theta \\
&= \cos\theta\frac{\partial u}{\partial x} + \sin\theta\frac{\partial u}{\partial y}.
\end{aligned}$$

$$\begin{aligned}
\frac{\partial u}{\partial \theta} &= \frac{\partial u}{\partial x}\frac{\partial x}{\partial \theta} + \frac{\partial u}{\partial y}\frac{\partial y}{\partial \theta} \\
&= \frac{\partial u}{\partial x}(-r\sin\theta) + \frac{\partial u}{\partial y}(r\cos\theta) \\
&= -r\sin\theta\frac{\partial u}{\partial x} + r\cos\theta\frac{\partial u}{\partial y}.
\end{aligned}$$

By calculating the second derivatives we get

$$\begin{aligned}\frac{\partial^2 u}{\partial r^2} &= \cos\theta\frac{\partial}{\partial r}\frac{\partial u}{\partial x} + \sin\theta\frac{\partial}{\partial r}\frac{\partial u}{\partial y}\\ &= \cos\theta\left(\frac{\partial}{\partial x}\frac{\partial u}{\partial x}\frac{\partial x}{\partial r} + \frac{\partial}{\partial y}\frac{\partial u}{\partial x}\frac{\partial y}{\partial r}\right) + \sin\theta\left(\frac{\partial}{\partial x}\frac{\partial u}{\partial y}\frac{\partial x}{\partial r} + \frac{\partial}{\partial y}\frac{\partial u}{\partial y}\frac{\partial y}{\partial r}\right)\\ &= \cos^2\theta\frac{\partial^2 u}{\partial x^2} + 2\cos\theta\sin\theta\frac{\partial^2 u}{\partial x\partial y} + \sin^2\theta\frac{\partial^2 u}{\partial y^2}.\end{aligned}$$

$$\begin{aligned}\frac{\partial^2 u}{\partial\theta^2} =& - r\cos\theta\frac{\partial u}{\partial x} - r\sin\theta\frac{\partial}{\partial\theta}\frac{\partial u}{\partial x} - r\sin\theta\frac{\partial u}{\partial y} + r\cos\theta\frac{\partial}{\partial\theta}\frac{\partial u}{\partial y}\\ =& - r\cos\theta\frac{\partial u}{\partial x} - r\sin\theta\left(\frac{\partial}{\partial x}\frac{\partial u}{\partial x}\frac{\partial x}{\partial\theta} + \frac{\partial}{\partial y}\frac{\partial u}{\partial x}\frac{\partial y}{\partial\theta}\right) - r\sin\theta\frac{\partial u}{\partial y}\\ &+ r\cos\theta\left(\frac{\partial}{\partial x}\frac{\partial u}{\partial y}\frac{\partial x}{\partial\theta} + \frac{\partial}{\partial y}\frac{\partial u}{\partial y}\frac{\partial y}{\partial\theta}\right)\\ =& - r\cos\theta\frac{\partial u}{\partial x} - r\sin\theta\left(\frac{\partial^2 u}{\partial x^2}(-r\sin\theta) + \frac{\partial^2 u}{\partial x\partial y}r\cos\theta\right)\\ &- r\sin\theta\frac{\partial u}{\partial y} + r\cos\theta\left(\frac{\partial^2 u}{\partial x\partial y}(-r\sin\theta) + \frac{\partial^2 u}{\partial y^2}r\cos\theta\right)\\ =& - r\left(\cos\theta\frac{\partial u}{\partial x} + \sin\theta\frac{\partial u}{\partial y}\right) + r^2\left(\sin^2\theta\frac{\partial^2 u}{\partial x^2} - 2\cos\theta\sin\theta\frac{\partial^2 u}{\partial x\partial y} + \cos^2\theta\frac{\partial^2 u}{\partial y^2}\right).\end{aligned}$$

Divide $\frac{\partial^2 u}{\partial\theta^2}$ by r^2 we get

$$\frac{1}{r^2}\frac{\partial^2 u}{\partial\theta^2} = -\frac{1}{r}\frac{\partial u}{\partial r} + \sin^2\theta\frac{\partial^2 u}{\partial x^2} - 2\cos\theta\sin\theta\frac{\partial^2 u}{\partial x\partial y} + \cos^2\theta\frac{\partial^2 u}{\partial y^2}.$$

Adding the $\frac{\partial^2 u}{\partial r^2}$ to it and using the obvious identity $cos^2(\theta) + sin^2(\theta) = 1$ we get

$$\frac{\partial^2 u}{\partial r^2} + \frac{1}{r^2}\frac{\partial^2 u}{\partial\theta^2} = -\frac{1}{r}\frac{\partial u}{\partial r} + \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}.$$

Then the Laplacian of u will be given by:

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = \frac{\partial^2 u}{\partial r^2} + \frac{1}{r}\frac{\partial u}{\partial r} + \frac{1}{r^2}\frac{\partial^2 u}{\partial\theta^2}.$$

So the Laplacian of Γ gives

$$\Delta\Gamma(r) = \Gamma''(r) + \frac{1}{r}\Gamma'(r).$$

Then we get the following equation by using the properties of δ.

$$\Gamma''(r) + \frac{1}{r}\Gamma'(r) = 0 \quad \text{for} \quad r > 0, \tag{3.10}$$

such that we have

$$\frac{\Gamma''(r)}{\Gamma'(r)} = \frac{-1}{r}.$$

By integrating both sides

$$\ln \Gamma' = \ln r + c_1,$$

then

$$\Gamma'(r) = \frac{c_1}{r}.$$

Consequently the solution of 3.10 will be given as:

$$\Gamma(r) = c_1 \ln r + c_2,$$

where c_1, c_2 are constants, taking $c_2 = 0$.
Using the divergence theorem with $F(x, y) = \nabla\Gamma(x, y)$, with the first property of delta distribution we get the following equality:

$$1 = \iint \delta(\mathbf{x}) d\mathbf{x} = \iint \Delta\Gamma(\mathbf{x}) d\mathbf{x} = \iint_{B(0,R)} \Delta\Gamma(\mathbf{x}) d\mathbf{x} = \int_{\partial B(0,R)} \mathbf{n} \cdot \nabla\Gamma(\mathbf{x}) \, \mathrm{d}S.$$

As Γ is a radial symmetric function, the directional derivative of it is the same as at $|\mathbf{x}| = R$

$$n \cdot \nabla\Gamma = \Gamma'(R) \quad \text{for } |\mathbf{x}| = R.$$

We know that $\Gamma'(R)$ is constant along the sphere then

$$\int_{\partial B(0,R)} \mathbf{n} \cdot \nabla\Gamma(\mathbf{x}) dS = \int_{\partial B(0,R)} \Gamma'(R) dS = 2\pi R\Gamma'(R)$$

consequently we get

$$1 = 2\pi R\Gamma'(R)$$

by solving this equality we get

$$\Gamma(R) = \frac{1}{2\pi} \ln(R).$$

Finally the function Γ will be given by:

$$\Gamma(\mathbf{x}) = \frac{1}{2\pi} \ln r, \quad \text{where } r = |\mathbf{x}| = \sqrt{x^2 + y^2}.$$

The function $\Gamma(\mathbf{x})$ is called the fundamental solution in $2D$ for the Laplacian.

The fundamental solution of the Laplace operator in $\mathbb{R}^n$ is given by

$$\Psi : \mathbb{R}^n \to \mathbb{R}, \quad \Psi(\mathbf{x}) = \begin{cases} -\frac{1}{2\pi} \ln |\mathbf{x}|, & n = 2 \\ -\frac{1}{(n-2)\alpha_n |\mathbf{x}|^{n-2}} & \text{for } n \geq 3, \end{cases} \tag{3.11}$$

where α_n denotes the volume of the unit ball in $\mathbb{R}^n$.

3.2 Green's function for bounded domains

Let us consider Ω bounded domain in $\mathbb{R}^2$ and the Poisson problem with Dirichlet boundary condition, and using the fundamental solution of the Laplacian. The Green's function with with Dirichlet boundary condition on $\partial\Omega$ is given by:

$$\begin{cases} \Delta G = \delta(\mathbf{x} - \mathbf{x}_0) & \mathbf{x} \in D \\ G(\mathbf{x}, \mathbf{x}_0) = 0 & \text{for } \mathbf{x} \in \partial D. \end{cases} \tag{3.12}$$

We can construct the function G as follows:

$$G(\mathbf{x}, \mathbf{x}_0) = \Gamma(\mathbf{x} - \mathbf{x}_0) + v(\mathbf{x}, \mathbf{x}_0),$$

for fixed $\mathbf{x}_0 \in \Omega$ in order to satisfy this formula the problem in 3.12, so we have to find v in the following Laplacian problem:

$$\begin{cases} \Delta v(\mathbf{x}, \mathbf{x}_0) = 0 \quad \mathbf{x} \in D \\ v(\mathbf{x}, \mathbf{x}_0) = -\Gamma(\mathbf{x} - \mathbf{x}_0), \quad \text{for } \mathbf{x} \in \partial D. \end{cases}$$

3.3 The Caldéron problem

The main goal of Caldéron inverse problem is to determine γ by knowing $\wedge_\gamma$. Formally, we have to determine the inverse ϕ^{-1} of the mapping

$$\phi : C^1(\overline{\Omega}) \to \mathcal{L}(H^{\frac{1}{2}}(\Omega), H^{-\frac{1}{2}}(\Omega)), \quad \phi : \gamma \longrightarrow \wedge_\gamma.$$

In the following derivation, we follow [12]. Using the notation Q_γ for the quadratic form associated to (3.4)

$$Q_\gamma(f) = \int_\Omega \gamma |\nabla u|^2 dx. \tag{3.13}$$

Polarizing this quadratic form Q_γ we obtain the following bilinear form:

$$Q_\gamma(f, g) = \int_\Omega \gamma \nabla u \cdot \nabla v.$$

Here u is the solution with the Dirichlet data f and v is the solution with the Dirichlet boundary condition equal to g.
The divergence theorem gives:

$$Q_\gamma(f, g) = \int_{\partial\Omega} g \Lambda_\gamma f$$

Then the Dirichlet integral form given in (3.13) will have the form:

$$Q_\gamma(f) = \int_{\partial\Omega} f \Lambda_\gamma f.$$

Therefore,

$$\Lambda_\gamma : H^{\frac{1}{2}}(\partial\Omega) \to H^{-\frac{1}{2}}(\partial\Omega)$$

is the unique self-adjoint operator associated with Q_γ with domain $H^{\frac{1}{2}}(\Omega)$. Hence, we can rewrite the inverse conductivity problem as a mapping study

$$\phi^{-1} : \gamma \longrightarrow Q_\gamma.$$

Caldéron gave the following special solution for the conductivity equation:

$$u(x) = e^{x\zeta}, \quad \zeta \in \mathbb{C}^n,$$

where ζ is fixed. Then

$$\frac{\partial u}{\partial x_j} = \zeta_j e^{x\zeta},$$

and similarly,

$$\frac{\partial^2 u}{\partial {x_j}^2} = \zeta_j \cdot \zeta_j e^{x\zeta}.$$

In this way,

$$\Delta u = 0 \implies \sum_{j=1}^{n} \zeta_j^2 = 0,$$

with

$$\zeta = \zeta_r + i\zeta_i \qquad\qquad \zeta_r, \zeta_i \in \mathbb{R}.$$

The Caldéron's problem is to invert DtN map $\gamma \mapsto N_\gamma$, and this map is not linear because of the variability of γ. Therefore, Caldéron studied a linearized problem near constant conductivities. Taking $\gamma = 1$, he focused on the map $\gamma \mapsto \phi_\gamma$ instead of $\gamma \mapsto N_\gamma$

$$\phi_\gamma(f) = \int_\Omega \gamma |\nabla u|^2 dx$$

3.4 Uniqueness of the DtN map

The uniqueness of the DtN map for the conductivity and for the Schrödinger problems were given for dimension $n \geq 3$ in the paper [17]. We fellow this work in the following presentation.

Theorem 3.4.1 (Sylvester and Uhlmann). *Let $\Omega \in \mathbb{R}^n$ be a bounded open domain with smooth boundary, where $n \geq 3$, and let γ_1 and γ_2 be two positive functions in $C(\bar{\Omega})$.*
If $\wedge_{\gamma_1} = \wedge_{\gamma_2}$ then $\gamma_1 = \gamma_2 \in \Omega$.
If $q_1, q_2 \in L^\infty(\omega)$ and $\gamma q_1 = \gamma q_2$ then $q_1 = q_2$.

Proof. First we prove that $\wedge_{q_1} = \wedge_{q_2}$ implies $q_1 = q_2$ in Ω for the Schrödinger equation. Using the Green's theorem we get

$$\int_\Omega (q_1 - q_2) u_1 u_2 \, dx = \int_{\partial\Omega} f_2 \wedge_{q_1} f_1 - f_1 \wedge_{q_2} f_2 \, dS.$$

Since $\wedge_{q_1}$ is a self adjoint operator, we also have

$$\int_\Omega (q_1 - q_2) u_1 u_2 \, dx = \int_{\partial\Omega} f_1 (\wedge_{q_1} - \wedge_{q_2}) f_2 \, dS.$$

Terefore, $\wedge_{q_1} = \wedge_{q_2}$ implies

$$\int_\Omega (q_1 - q_2) u_1 u_2 \, dx = 0.$$

Let now $u_i(x) = e^{x\zeta_i}, \ i = 1, 2$ be special solutions to $L_q u_i = 0$ with

$$\zeta_1 = \frac{\eta}{2} + i\frac{k}{2}$$

$$\zeta_2 = -\frac{\eta}{2} + i\frac{k}{2},$$

where $\eta, k \in \mathbb{R}^n$ with $\eta \cdot k = 0$. Substituting u_1, u_2 into (2.1) gives

$$\int_\Omega e^{x.k_i} (q_1 - q_2) \, dx = 0,$$

then $q_1 = q_2$. We know that if $\wedge_{\gamma_1} = \wedge_{\gamma_2}$ then $\wedge_{q_1} = \wedge_{q_2}$ with $q_i = \frac{\Delta\sqrt{\gamma_i}}{\sqrt{\gamma_i}}$. Now it is easy to prove that $q_1 = q_2$ implies $\gamma_1 = \gamma_2$. □

Chapter 4

Approximation of the DtN map using neural networks

An artificial neural network (or shortly neural network) is a structured directed graph representing a connection between its vertices. The motivation of the construction is the complex structure of the brain or a real-life neural system. The vertices are grouped into consecutive layers including a specific input and output one. Setting an appropriate structure and weights on the edges, a set of algorithms can be obtained that attempt to recognize the underlying relationships among data. In this way, neural networks became very powerful computational tools. They have revolutionized fields such as image, text, speech recognition, computer vision and more. Beyond the linear effect of the above weights, nonlinear functions are also included in the construction. Also, high-dimensional inputs in these fields requires statistical approaches.

Indeed, the computational algorithm is a kind of optimization, which is also called *learning* or *machine learning*. Basically, it has two main types:

1. **Supervised learning** involves training the machine on data that is already given with the correct answer. A supervised learning algorithm learns from labeled training data, so that it can predict outcomes for unexpected data. The model problem of supervised learning can be formulated mathematically as follows: given a set

$$S = \{(\boldsymbol{x}_i, \boldsymbol{y}_i = f^*(\boldsymbol{x}_i)), i \in \{1, 2, 3, ..., n\}\}$$

   we should try to approximate f^* as accurately as possible. Taking continuous values of f^* is called a regression problem, where taking discrete values is called a classification problem.

   (a) **Classification** model use an algorithm to categorize test data accurately into specific categories.

   (b) **Regression** model is an algorithm used to discover the relationships between dependent and independent variables. A regression model can be used to predict numerical values based on different data points. For a visualisation for this problems, see figure 4.1

. Considering a PDE or an ODE and taking into consideration Dirichlet boundary conditions or other type of conditions can be recognized as the given data. Taking a large set of them with the corresponding solutions can be used to train the model. Based on this, we try to predict the solution at "mesh" or random points from a domain Ω in $\mathbb{R}^n$.

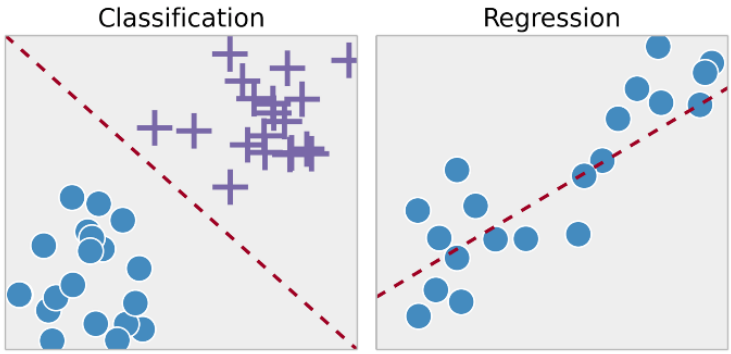


Figure 4.1: Types of supervised machine learning [11]

2. **Unsupervised learning** method uses machine learning algorithms to analyze unlabeled data sets and cluster them accordingly. Algorithms like these are capable of discovering hidden patterns without the need for humans to be involved. There are three main tasks that unsupervised learning models can perform: clustering, association and dimension reduction.

   (a) **Clustering** The idea behind clustering is to group unlabeled data according to what they have in common, or what they differ from each other.

   (b) **The association** method is another type of unsupervised learning method that employs a variety of rules to determine the relationship between variables within a dataset.

   (c) **Dimension reduction** when there is an excessive number of dimensions in a dataset, dimension reduction is used to reduce the number of dimensions. As a result, the number of data inputs is reduced to a manageable size while maintaining the integrity of the data. For more details see these articles [11] and [9].

4.1 Model architecture of neural networks

In this thesis, we are using neural networks based on supervised learning. We will give a neural network by the following mathematical map: $\mathcal{NN} : \mathbb{R}^{n_x} \mapsto \mathbb{R}^{n_y}$ where the first layer is presented as a vector in $\mathbb{R}^{n_x}$, and the last layer is a vector in $\mathbb{R}^{n_y}$. The dimensions n_x and n_y present the number of neurons in the input and the output layer, respectively.

Layers consist of containers (neurons) which receive a weighted input, transform it with a set of mostly non-linear functions, and then pass these values as output to the next layer. From hereafter layers are described as vectors where the first layer is a vector of dimension n_x and the last layer is a vector of dimension n_y.
The first layer of a NN is called the input layer, followed by some hidden layers, and the last one is called the output layer. Each two consecutive layers are connected to each other by the so called the weights and biases.

Dense layer can be defined as a layer that is deeply connected to its preceding layer, meaning each neuron of that layer is connected to every neuron of that layer. It can be understood mathematically as vectors denoted by $h_1, h_2, ..., h_{n-1}$ such that

$$h_i = f(W * h_{i-1} + b) \quad h_i \in \mathbb{R}^{n_i}, \quad i \in \{1, 2, ...n\}$$

h_o and h_n are the input and output layers respectively. The function f is defined as the activation function and W is the weight, and b presents the bias term.

Convolutional layer convolution is the simple process of applying a filter to an input to activate it. When a filter is applied repeatedly to an input, it generates a map of activation called a feature map, indicating the locations and strengths of detected features this layer is often used in image recognition. However, this layer may be helpful for us if we use the one dimensional case. As we know, a convolutional layer contains several hyperparameters, such as how many filters we have, how large the filters are, and the stride. Filters are used to learn patterns, and each filter learns a different pattern.

A filter (Kernel) consists of a matrix of weights that is multiplied by the input parts. A map of activation is result of the multiplications of these results.

Remark 2. *Using more than one filter might be helpful sometimes if we want to get different patterns on one layer.*

Kernel size refers to the size of each kernel, which is composed of weights that are tuned for better detection of certain features. In other words it can defined as how much the input data is processed during each convolution.

Stride is the amount by which the kernel shifts or moves each time it is computed.

Padding is a way to control the shape of our output, when using convolutional layer. These open sources use three types of padding, which we can be briefly described as follows:

1. **Same** by adding some zeros to the input, we make the output the same length as the input by going through all input and producing the same length in the output.

2. **Valid** no padding.

3. **Causal** results in causal (dilated) convolutions.

Remark 3. *The locally connected layer is quite the same as the Convolution layer, but there is only one difference is that the locally connected layer applies different sets of filters to different patches of input, so that weights are not shared.*

Example 4.1.1. *We gain a better understanding of how hyperparameters work through the following example:*

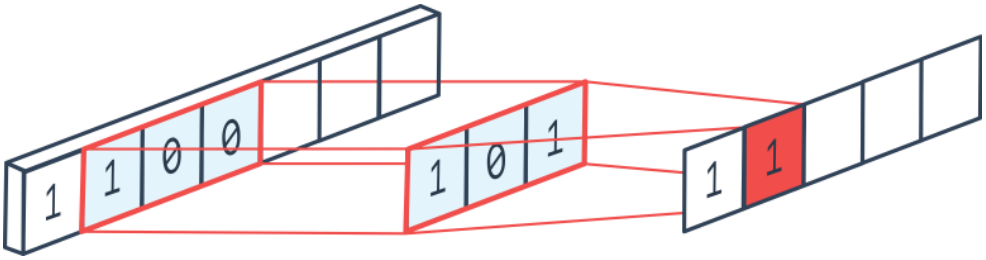


Figure 4.2: Example of A 1D convolution with a kernel sized 3 and stride 1 along with valid padding.[15]

Weights in the neural networks are defined as the parameters which transform the data in a specific layer to the next one.

Taking the data $\mathbf{x}_j \in \mathbb{R}^{d_j}$ in layer j, and the data $\mathbf{x}_{j+1} \in \mathbb{R}^{d_{j+1}}$ in layer $j+1$, the set of weights correspond to a matrix $\mathbb{R}^{d_j \times d_{j+1}}$.

Bias is an additional constant term to the output layer of every two consecutive connected layers, its role is performing a shift. This helps us to control and give good estimation to the real output data.

Example 4.1.2. *Consider the following Neural network with two layers. One of the consists of three neurons and the other one consists of two. They are given by* $\mathbf{x}_j = (x_{j_1}, x_{j_2}, x_{j_3})$ *and* $\mathbf{x_{j+1}} = (x_{j+1,1}, x_{j+1,2})$ *and the neural network can be represented as*

$$\begin{aligned}\begin{pmatrix} x_{j+1,1} \\ x_{j+1,2} \end{pmatrix} &= \begin{pmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{pmatrix} \begin{pmatrix} x_{j_1} \\ x_{j_2} \\ x_{j_3} \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} \\ &= \begin{pmatrix} w_{11} * x_{j_1} + w_{12} * x_{j_2} + w_{13} * x_{j_1} + b_1 \\ w_{21} * x_{j_1} + w_{22} * x_{j_2} + w_{23} * x_{j_1} + b_2 \end{pmatrix}.\end{aligned}$$

Here the weights are given as parametric matrix W *with the entries* w_{jk}*, and the parametric bias is given by the vector* (b_1, b_2)*.*

In this way, we have described feedforward neural networks, where the connections between nodes do not form a cycle. Recurrent neural networks on the other hand can consist of pathways that recur. The feed forward model involves processing information only in one direction, making it the simplest form of neural networks. Despite passing through multiple hidden nodes, the data always moves forward and never backward. The structure of a neural network is shown in the following figure.

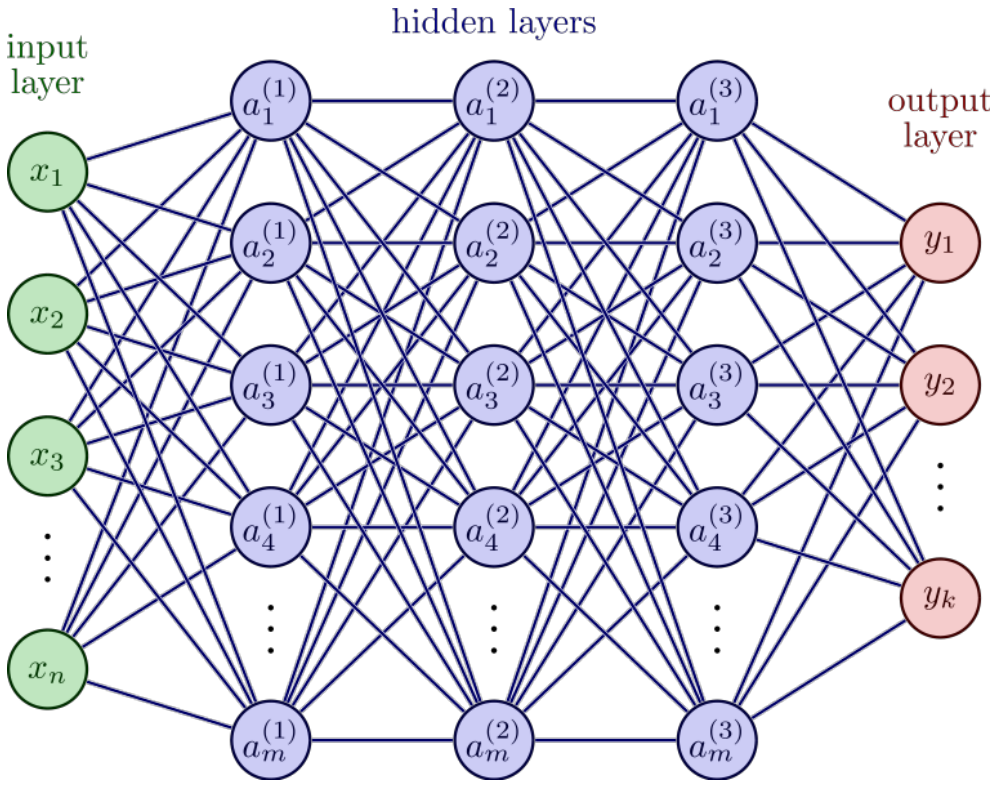


Figure 4.3: Structure of a neural network with 5 fully-connected layers

4.1.1 Activation function

An activation function determines in real neural networks whether or not the neuron should be activated. In other words an activation function determines whether the neuron's input to the network is significant or not in the prediction process.
Accordingly, its mathematical model is a function $g : \mathbb{R} \to \mathbb{R}$ such that the entire mapping between two consecutive layers can be given as
$\mathbf{x_{j+1}} = g(W\mathbf{x}_j + b)$. We mention some types of an activation function.

1. **Sigmoid** function is defined by

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

2. **Tanh** function is given by the formula

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

As we are implementing a neural network to obtain the Dirichlet to Neumann data, we must be careful not to lose some negative data. In this way, the activation function **tanh** can be helpful for us to adjust the output data.

3. **The ReLU** (Rectified Linear Unit) function is defined by the following formula

$$f(x) = max(0, x).$$

The activated neuron in ReLU is the neuron whose output is more than 0.

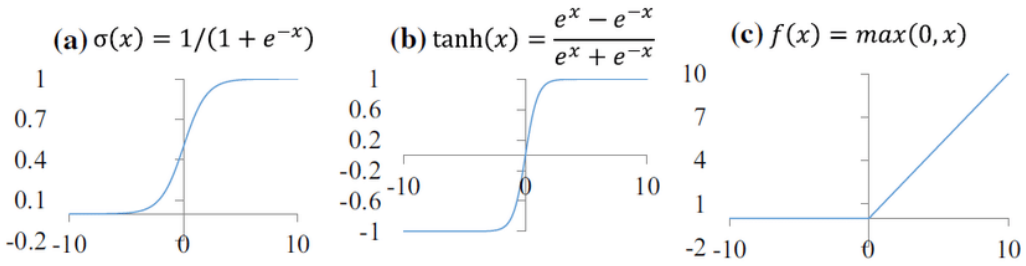


Figure 4.4: The activation functions (a) Sigmoid, (b) tanh, (c) ReLU.

4.1.2 Training, validating , testing datasets

For more detailed discussion, we refer to [2]. We introduce briefly the train, test and validation data sets. The purpose of data splitting is to prevent overfitting. This is an instance in which a machine learning model is able to fit its training data too well and is unable to fit unseen data reliability.

The training set is used to fit the parameters based on examples that have been collected throughout the learning process. Supervised learning algorithms to determine the best combination of variables that will generate a reliable prediction. Our objective is to produce a trained model that generalizes well when faced with new, unknown datasets. Mathematically, the training data can be considered as set of of pairs (x, y) from a data set.

A validation set can be considered again as a set of pairs $(\tilde{x}, \tilde{y})$ taken from a dataset. It is used to evaluate the fit of a model against a training dataset while tuning and adjusting the hyperparameters. It is sometimes also called the development set, an example of a hyperparameter for artificial neural networks includes the number of hidden units in each layer.

Testing set is a sample of data used to evaluate the accuracy or the error function of a final model (which was built by training and validation data sets) fit in light of the training dataset and measure its performance. To do that the final model is used to predict a test set and compare the result of predictions with the true data set. The testing data set is distributed in a similar manner

to the training data set. If we have the trained neural network, we compare the prediction $\mathcal{NN}(x)$ with y.

Properties

1. The test dataset is typically used to evaluate the final model selected during the validation process, in situations where both validation and test datasets are used.

2. The test data set may only assess the model once if the original data set is divided into two subsets (training data and test data)

The below figure shows how the model is processed with splitting the data

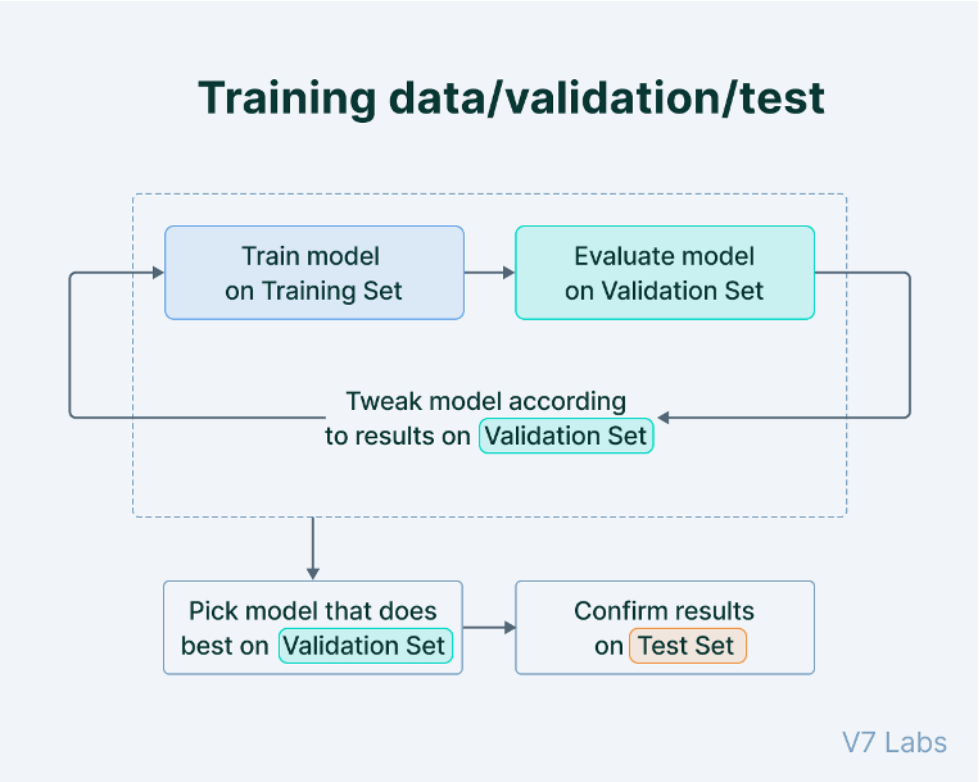


Figure 4.5: The Processes which are taken to get a neural network model [7].

4.1.3 Error measurement functions

In the following two paragraphs, we are going to introduce two fundamental estimators to measure the performance of the model.

The loss function is used to evaluate the performance of a machine learning algorithm with regards to the featured data set. In other words the loss function is a measure of how well the model predicts the expected outcome, the loss function is used commonly for supervised learning.

Mean square error shortly denoted by MSE, it is calculated as the average of the squared differences between the predicted and actual values, MSE is really known for regression model in supervised learning. Its mathematical formula is given in this way

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}\left(\hat{Y}_i - Y_i\right)^2,$$

such that $\hat{Y}_i \in \mathbb{R}$ are the predicted values where $\hat{Y} = (\hat{Y}_1, \hat{Y}_2, ..., \hat{Y}_n)$ is the output of the NN, and $Y_i \in \mathbb{R}$ are the actual values.

The accuracy is used mainly in a classification machine learning algorithm and it defined as a measure of how often it correctly classifies a data point. In other words the accuracy measurement is the percentage of correctly predicted data points out of all the data points. As a more formal definition, it is determined by dividing the number of true positives and true negatives by the number of true positives, true negatives, false positives, and false negatives, we say a data point is a true positive or true negative if the algorithm correctly classified it, by contrast, a false positive or false negative is a data point that is incorrectly classified by the algorithm. The formula of the accuracy is given in the following way

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}.$$

In the case of binary classification, accuracy can also be expressed in terms of positives and negatives as follows:

$$\textit{Accuracy} = \frac{\text{True positive + True negative}}{\text{True positive + False positive + True negative + False negative}}.$$

The accuracy can be applied to the classification supervised machine learning algorithm it presents the percentage of getting the correct classification. However, it cannot be applied to our model due to the fact that we are attempting to predict input data and measure how close our prediction is to the actual data. The loss function serves this purpose.

In the neural networks, each attribute has important role, the weight and bias are approximated to get good estimation. Moreover, the activation function has to be chosen carefully.

4.1.4 Optimization strategies

The main algorithm step is the approximation of the weights and biases based on the loss function. We discuss some principles and strategies with consecutive layers.

Definition 4.1.1 (Gradient descent). *Gradient descent is an iterative optimization algorithm to find the minimum of a function. In machine learning, the*

objective function needed to minimize is the loss function, so taking the following MSE function

$$L(W) = \frac{1}{n}\sum_{i=1}^{n}\left(\hat{Y}_W^{(i)} - Y^{(i)}\right)^2,$$

the gradient descent approximates the gradient of $L(W)$ at a single sample, and it takes the following formula to find the minimum

$$W = W - \alpha\nabla L(W) = W - \frac{\alpha}{n}\sum_{i=1}^{n}\nabla L_i(W).$$

Where $L_i(W)$ is the value of the loss function at $i-th$ sample (example), W is then weight matrix and α is the learning rate.

Gradient descent processes in deep learning the following steps are taken in order to get the minimum loss :

1. **Forward propagation** is the input data which flows from the input layer forward through the network. Every hidden layer accepts the input data, processes it based on the activation function, and then feeds the processed data as the input data to the next hidden layer.
2. **Estimate error** by calculating the loss function.
3. **A backward propagation** process propagates errors backward and updates hidden layer weights accordingly.
4. **Adjust weights and biases.**

Evaluation of the gradients may be costly for calculating the sum gradient especially if we use large amount of training data then gradient descent will be very computationally expensive. **Types of Gradient Descent:**

1. Batch Gradient Descent
2. Stochastic Gradient Descent
3. Mini-batch Gradient Descent

The important optimization method for us is the second one.

Stochastic gradient descent The word Stochastic refers to random so this method basically is based on taking a batch of one random sample this batch is randomly selected for performing an iteration of gradient descent and finding the of minimum of this sample. So instead of summing all the gradients of the cost functions of all the samples (examples), SGD finds the gradient of one example at each iteration. Despite the fact that this method can be noisier and more chaotic than gradient descent, because we take a random example to find the minimum every time and more iterations are needed than in a typical gradient descent method To reach the minima. But this does not matter as long as we are reaching the minimum in a short period of time and we are taking computationally much less expensive than gradient descent.

Adaptive Moment Estimation is denoted shortly by Adam. The Adam optimization algorithm uses repeated cycles of adaptive moment estimation to provide more efficient neural network weights. Adam extends stochastic gradient descent to solve nonconvex problems faster and with fewer resources than most optimization programs. During training, stochastic gradient descent maintains the same learning rate α for all weight updates where Adam method computes individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients. The first and the second moments of gradients are given in the following equations:

$$m_t = \beta_1 * m_t + (1 - \beta_1) * \left(\frac{\partial L}{\partial w_t}\right),$$

and

$$v_t = \beta_2 * v_t + (1 - \beta_2) * \left(\frac{\partial L}{\partial w_t}\right)^2.$$

The Adam equations is constructed by combining the momentum equation with the Root Mean Square Propagation (RMSP) equation. Assuming the hyperparameters β_1 and β_2 are close to zero. Then in the following section, we will introduce the Adam algorithm.

Algorithm 1 Adam algorithm

while $w(t)$ not converged **do**
$m_t = \beta_1 * m_t + (1 - \beta_1) * \left(\frac{\partial L}{\partial w_t}\right)$
$v_t = \beta_2 * v_t + (1 - \beta_2) * \left(\frac{\partial L}{\partial w_t}\right)^2$
$\hat{m}_t = m_t \div (1 - \beta_1^t)$
$\hat{v}_t = v_t \div (1 - \beta_2^t)$
$w_t = w(t-1) - \alpha * \left(\hat{m}_t / \sqrt{(\hat{v}_t}\right) + e\Big)$
end while
return $w(t)$

Weights at time t = w_t
α_t = learning rate (Hyperparameter)
e = constant
$m_(t)$,$v(t)$ are aggregate of gradients at time t and aggregate of gradients of gradients at time t.

We refer to [1],[4] and [6] for more details about the SGD and Adam methods.

Learning rate is an important hyperparameter, it is defining as the number of steps taken to reach the minimum. This number is usually small, and it is assessed based on how the cost function behaves to achieve the minimum. The learning rate should be carefully chosen as if we give a very small learning rate, the optimization method will take a long time to achieve the minimum, whereas if we give a very large learning rate, we may overshoot the minimum.

Batch Is a set of samples sent through a neural neetwork in a single pass. Here a sample means the values of ψ_y for a fixed y.

Epochs The number of times the entire training set is sent through the neural network.

Remark 4. *1. An epoch has one or more batches.*

2. Epochs sizes can be higher to achieve better loss function.

3. Batch size and epoch size are hyperparameters that can tuned to improve model accuracy.

The main idea of our investigation is to apply fundamental solutions and expand our model to find the DtN data for other problems.

The following feed forward neural network structure with two dense layers summarizes how our first model looks like:

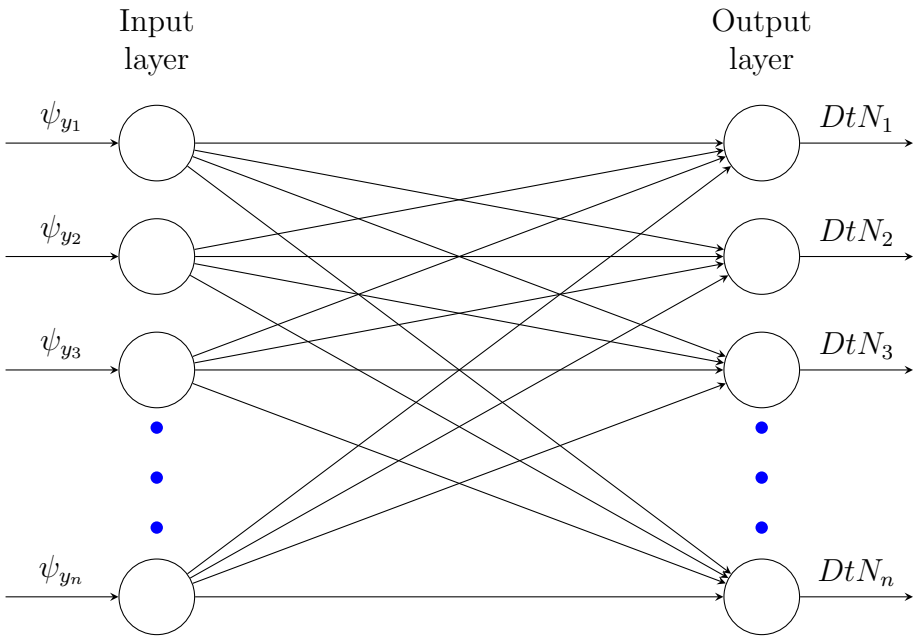


Figure 4.6: A feed forward neural network with two layers

4.2 Implementation and numerical results

Let us consider the Laplacian equation. In order to predict the DtN for some points on a boundary. We will construct a feed forward neural network, and since DtN is linear for the Laplace problem, we expect the model to learn from two layers. We will start our NN with two layers and then expand a bit as needed to compare its performance with the number of layers we use. We will start our NN with dense layers and then we try to include convectional and locally connected layers. The main open source libraries which used for the neural networks are Keras and Pytorch, and we will compare their performance

and use the one which give us the best result taking into consideration the running time and the loss function.

4.3 Principles for the implementation

To present the implementation of our problem, we will give structure of our computer code with full details. Within Python, we make use of the Keras library to create neural networks easily.
Using the command *sequential*, we can easily insert more layers into our neural network. We will use supervised learning techniques specifically regression type to feed the model some samples for the input and output layers and allow it to learn by itself. According to the real Dirichlet to Neumann operator, we will use two different approaches to get the Neumann data.
The first approach is to take the Dirichlet data at certain boundary points as input in an input layer. Likewise, the Neumann data at the same points will serve as an output data. We will try to use both a shallow neural network with a single layer and also more hidden layers. Since the Dirichlet to Neumann operator is linear, in all cases, we apply a linear activation function. Also, no bias terms are used between the layers. Then we try to include the bias as well. The first thing we're going to do is train the model, then we'll validate it with some other data which is what we get by using the same distribution with training data $\psi_y(x)$ for fixed points y. Using this training model, we can find parameters that predict Neumann data on a boundary with points we have never used in the model before, but first we need to find out how good our model is doing. The MSE loss function provides us with a way of measuring how well the model works, by comparing the predicted values to the actual values. We should have a better model if we get this loss function as close to zero as possible. The second approach is to take Dirichlet boundary data as input data and the values of the distribution function at the $x + h\nu$ for some negative h close to zero, and for ν is the outer normal vector. The reason why we take negative values of h because we need to set the points inside the domain Ω. We will use the same procedure, we used for the first approach of building the neural network by one dense layer, and then expand the number of the hidden layers to compare the results we take the same amount of train data and validate data even the same measure (MSE) to evaluate our model. This approach is different from the first because in this estimation we are trying to predict the value of the points inside the domain (the solution u at $x + h\nu$), then using the following formula of the directional derivative to approximate the Neumann data:

$$\partial_\nu f(\mathbf{x}) = \lim_{h \to 0} \frac{f(\mathbf{x} + h\nu) - f(\mathbf{x})}{h}.$$

In this way, the neural network will deliver a fast computational tool to transfer from Dirichlet data to Neumann data. However, the question that arises is, how can we find parameters that minimize loss?.
As a result, we have to solve an optimization problem in order to make the

loss function as close to zero as possible. We apply the Adam or Stochastic gradient descent (SGD) method to solve the problem.

4.4 Details of the implementation

Let us consider the Laplace problem $\Delta u = 0$ with Dirichlet boundary conditions. We use the concept of the fundamental solutions given with

$$\psi_{\mathbf{y}_j} : \mathbb{R}^2 \backslash \{\mathbf{y}_j\} \to \mathbb{R}, \quad \psi_{\mathbf{y}_j}(\mathbf{x}) = -\frac{1}{2\pi} \ln |\mathbf{x} - \mathbf{y}_j| \,.$$

In the first model, we take a concave domain Ω in $\mathbb{R}^2$. The Dirichlet to Neumann operator will be approximated in N points $\mathbf{x}_i$ on the boundary. Let us denote these points by $\partial\Omega_{\mathbf{x}_i}$. Using the distribution $\psi_{\mathbf{y}_j}$ with approximately M points, such that the points $\mathbf{y}_j \in \Omega^c$ outside the domain and its directional derivative responding to the outer normal vector $\boldsymbol{\nu}$, so we need first to compute the values of $\psi_{\mathbf{y}_j}(\mathbf{x})$ for all points $\mathbf{x} \in \partial\Omega_{\mathbf{x}_i}$, then compute the directional derivative of $\psi_{\mathbf{y}_j}$ using the following formula:

$$\partial_{\boldsymbol{\nu}} \psi_{y_j} = \nabla \psi_{\mathbf{y}_j} \cdot \boldsymbol{\nu} \tag{4.1}$$

We will eventually get two matrices of size $M \times N$, one of which represents input data, the other representing output data. These matrices are presented below for input and output data respectively:

$$Input = \begin{bmatrix} \psi_{\mathbf{y}_1}(\mathbf{x}_1) & \cdots & \psi_{\mathbf{y}_1}(\mathbf{x}_N) \\ \vdots & \ddots & \vdots \\ \psi_{\mathbf{y}_M}(\mathbf{x}_1) & \dots & \psi_{\mathbf{y}_M}(\mathbf{x}_N) \end{bmatrix}$$

$$Output = \begin{bmatrix} \partial_{\boldsymbol{\nu}}\psi_{\mathbf{y}_1}(\mathbf{x}_1) & \cdots & \partial_{\boldsymbol{\nu}}\psi_{\mathbf{y}_1}(\mathbf{x}_N) \\ \vdots & \ddots & \vdots \\ \partial_{\boldsymbol{\nu}}\psi_{\mathbf{y}_M}(\mathbf{x}_1) & \dots & \partial_{\boldsymbol{\nu}}\psi_{\mathbf{y}_M}(\mathbf{x}_N) \end{bmatrix}$$

Within our first model we have we have implemented two approaches.

Approach 01:

- Number of layers: only one input and and output layer.
- Size of the layers: 89 for both.
- Connectivity between the layers: dense.
- Activation function: linear.
- Bias term: no.
- Optimization algorithm: ADAM.

In this approach we had to use 89×89 parameters which need at least 9000 training data. We have tried to reduce the computing time by taking less parameters this led to the second approach.

Approach 02:

- Number of layers: only one input and and output layer.
- Size of the layers: expand the size of the first layer to be oscillated, we will have to add three more neurons to the beginning of the layer and three more to the end, which will make a total of 95 neurons. The output layer will have 89 neurons.
- Connectivity between the layers: locally connected.
- kernel size: 7.
- The activation function, bias and the optimization method the same as the first approach.

Second model

This model uses the same structure as the approach 01 from the first model with only two layers, but the output data we're trying to predict is $f(x + h\boldsymbol{\nu})$, we are going to try multiple values for h and we have to make sure that these points are inside the domain Ω. We are taking the same amount of train and test data we take the same optimization method and the same activation function, the only difference is that the output data are fed from the following matrix:

$$\text{Output}_2 = \begin{bmatrix} \psi_{\mathbf{y}_1}(\mathbf{x}_1 + h\boldsymbol{\nu}) & \cdots & \psi_{\mathbf{y}_1}(\mathbf{x}_N + h\boldsymbol{\nu}) \\ \vdots & \ddots & \vdots \\ \psi_{\mathbf{y}_M}(\mathbf{x}_1 + h\boldsymbol{\nu}) & \dots & \psi_{\mathbf{y}_M}(\mathbf{x}_N + h\boldsymbol{\nu}) \end{bmatrix}$$

Then we use the formula (4.1) to approximate the Neumann data, eventually we use the same solution u for testing the model and predict the solution u at the points $\mathbf{x}_i + h\boldsymbol{\nu}$ then we use the formula of the directional derivative to estimate the Neumann data.

4.4.1 Low-rank approximation

Our neural network-based approach delivers a full matrix. This corresponds to a linear function transforming the Dirichlet boundary data into the Neumann data. Whenever, in practice, this completely describes the conductivity properties of the domain Ω, it contains a huge number of parameters. In case of inverse problems, e.g., for the electrical impedance tomography, we need this parameters as input data. Therefore, to reduce the computational complexity of those problems, it would be highly desirable to reduce their number.

In other words, we should try to reduce the dimension of the data without a significant loss in the accuracy of our approximation of the Dirichlet to Neumann problem. This motivated us to perform also low-rank approximations. In practice, we can easily implement it in the framework of the neural networks.

We only have to insert an extra layer of length L between the input and output layers. Of course, we need a dense layer for minimizing the loss of information. The neural network in this case corresponds to a product of matrices of size $89 \times L$ and $L \times 89$, which altogether results again a matrix of size 89×89. At the same time, its rank is only L and is automatically decomposed as a product of the above matrices.

4.5 Results of the numerical experiments

On a simple laptop, we carried out our experiments using the Keras deep learning library, which is an open source package in Python. Besides using this package to build the neural network and fit the problem, we also use Numpy for vectors and matrices, and Matplotlib for visualization of the domain, loss function and DtN data. We get the domain Ω in the figure 2 by taking N=89, M=10000.

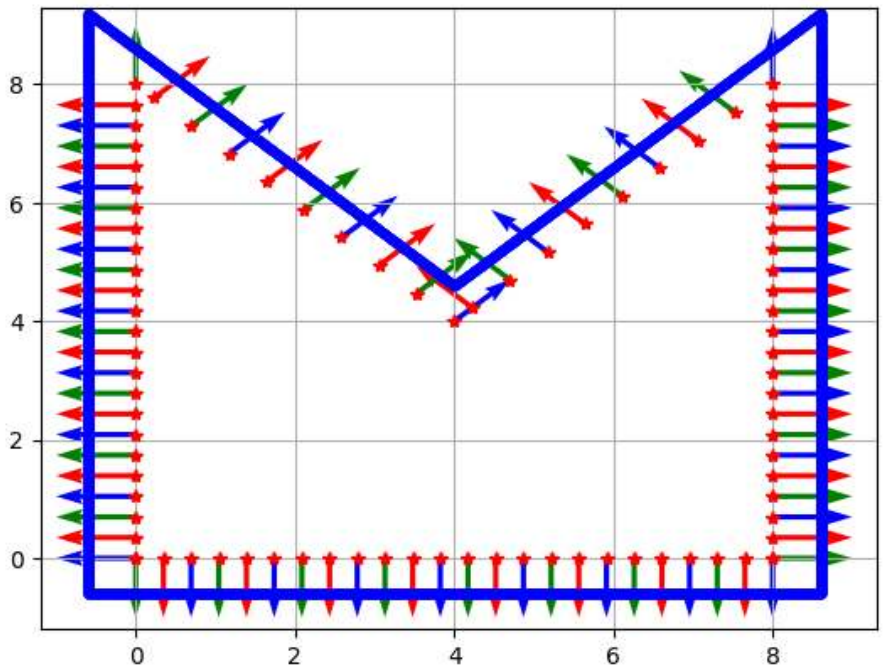
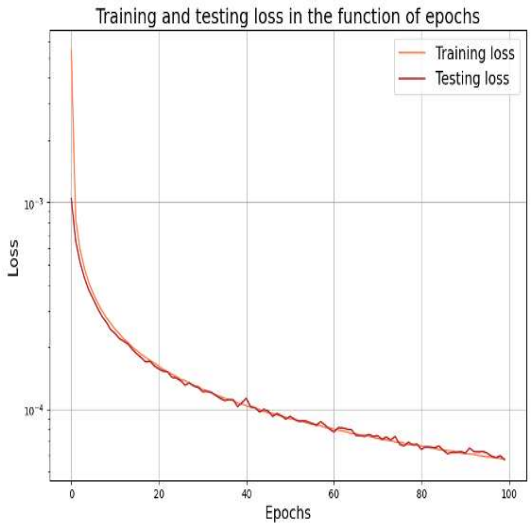
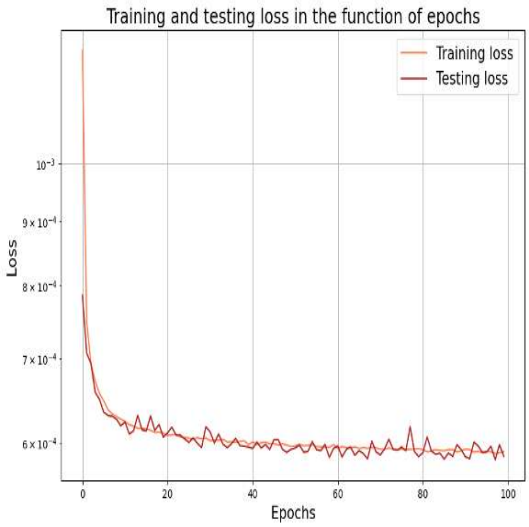


Figure 4.7: The computational domain with the boundary points (89 points in red) and the outer points (10000 points in blue).

The following figures show the mean squared error in all the methods we used in order to find the Neumann data based on approximating the DtN map. We modified the number of neurons in the hidden layers to determine which model is the best to maintain.

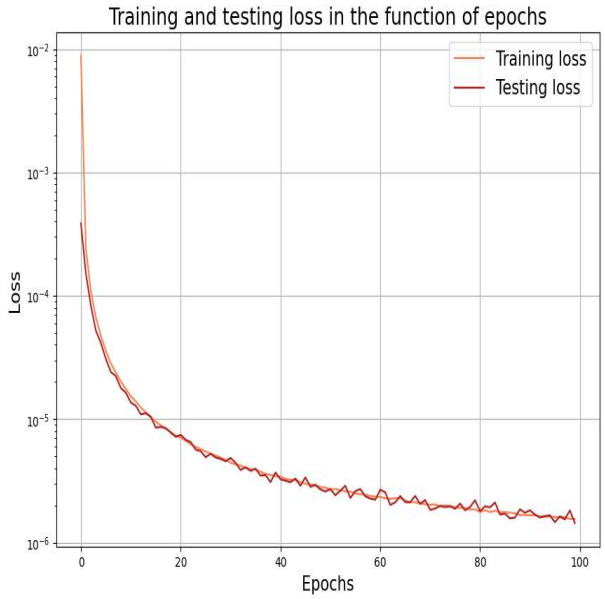


(a) The NN with one dense layer (**Approach 01**)

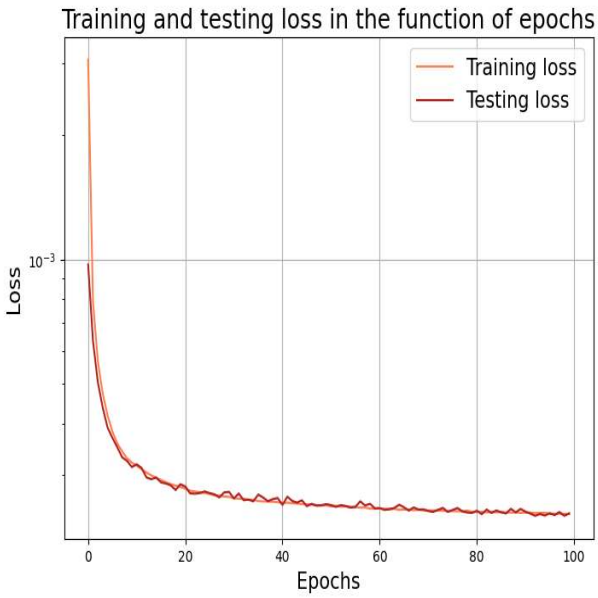


(b) The NN with one locally connected layer (**Approach 02**)

Figure 4.8: : Training (orange) and testing (red) loss over 100 epochs in the first model.

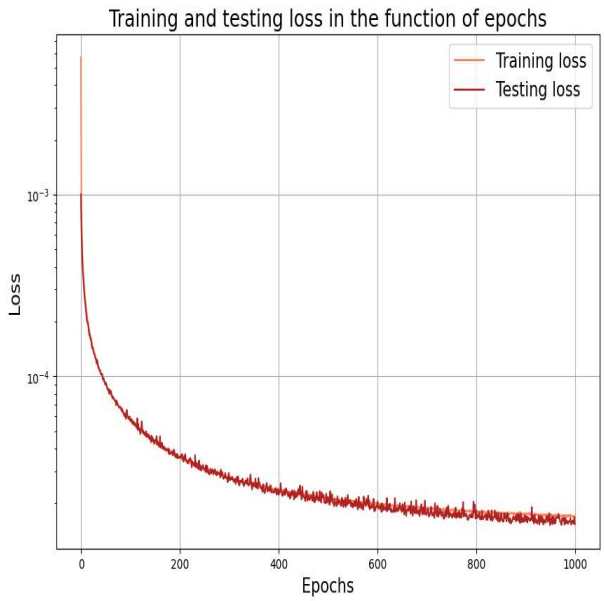


(a) The NN with one dense layer in the second model (**Approach 01**)

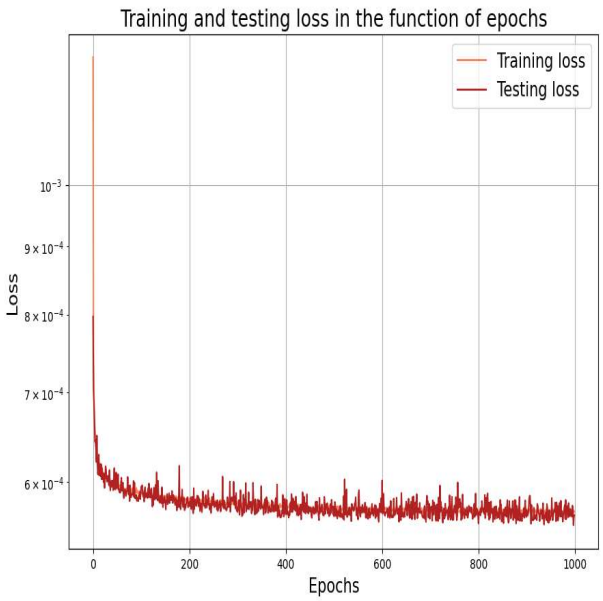


(b) The NN with two dense layers and (**L=20**)

Figure 4.9: : Training (orange) and testing (red) loss over 100 epochs in the second model and in the Low-rank approximation.

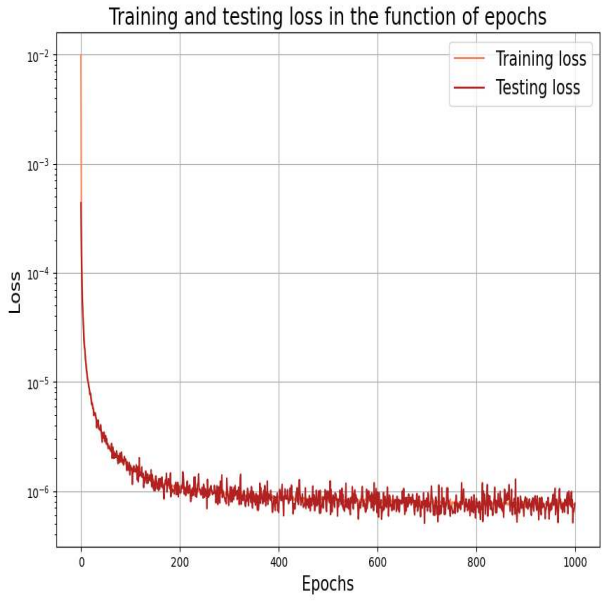


(a) The NN with one dense layer (**Approach 01**)

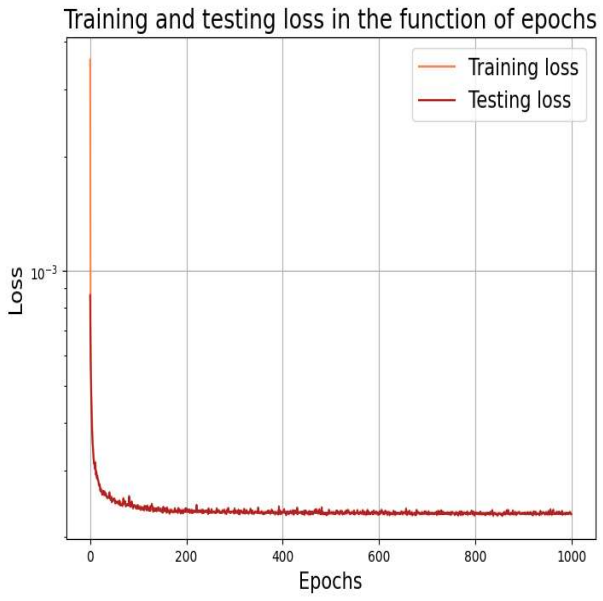


(b) The NN with one locally connected layer (**Approach 02**)

Figure 4.10: : Training (orange) and testing (red) loss over 1000 epochs in the first approach.



(a) The NN with one dense layer (**Second model**)



(b) The NN with two dense layers and (**L=20**)

Figure 4.11: : Training (orange) and testing (red) loss over 1000 epochs in the second model and in the Low-rank approximation.

We trained the neural network over 100 epochs. Besides Adam, Stochastic gradient descent (SGD) and other optimization methods were used to optimize

the parameters, but Adam performed the best, we took the value of the hyper parameters for all approaches as follows:

1. Learning rate with 0.001.

2. Batch size equal to 32 (Default).

There are 89 neurons in the input and output layers since that is the number of columns. The neural network was trained by 9000 rows which presents 90% of the data and validated by 1000 rows(the rest of the data) from the Input and output matrices.

In our experiment, we found that dense layers are more efficient than other types of layers. Despite both losses decreasing, we used NN with Convolutional layers whose loss functions were underfitted, but at some point there was a constant loss. The result of using locally connected layers, which are a special case of convolutional layers, can be seen in Figure 4.10 (b). Since this type of layer only has valid padding, we had to increase the number of neurons in the input of this model. We added other layers to the NN, the result improved but only slightly. Convolutional layers were used to reduce parameters.
By using the Low rank approximation method, we added more dense layers to the model in Figure 4.11 (b). However, the first approach remains the most accurate.
In the Figure 4.11 (a), we present the loss function in second model with $h = -0.2$. We can observe that the model learned well over 1000 epochs with only one dense layer. We can observe also that the training and testing loss are decreasing similarly. This is an indicator of a good neural network.
For a better value experiment, we must test our neural network against unseen solutions to show that it finds the DtN map for every Laplace problem with Dirichlet boundary condition.
The NN is tested by the solution $u(x, y) = (x-1)^2 - (y+3)^2$ with its Dirichlet data on the boundary given as an input data, and $\partial_\nu u = \langle \nabla f, \nu \rangle$ as Neumann output data.
New data predictions take no more than two seconds, so we will predict the DtN data for all models. In the next figure, we are showing how well the Second model performed when we tried to predict the points inside the domain. Starting from the top-left, the points are numbered from 1 to 89 on the axis, going through the boundary to the nearest point to the first one.

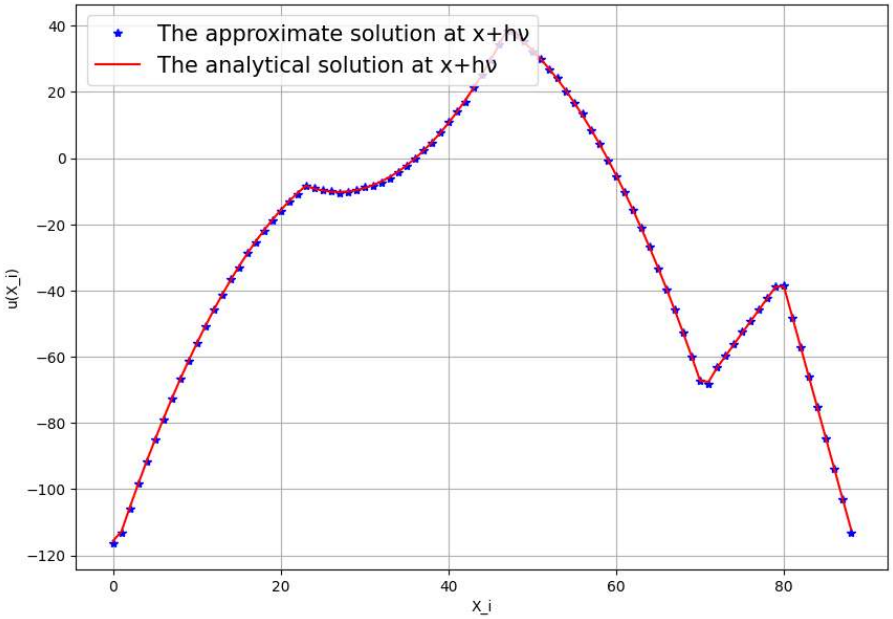


Figure 4.12: The exact and the approximated solution u at the points $x + h\nu$ with $h = -0.2$

The following table shows the number of the trainable parameters in each model.

Approach 01	**Approach 02**	**Second model**	**Low-rank approximation L=20**
7,921	801	7,921	3,560

Table 4.1: The number of parameters for each model

The following table shows MSE for all different models using different number of epochs.

Epochs	**Approach 01**	**Approach 02**	**Second model**	**Low-rank approximation**
50	1.63	337.646	0.706	1.887
100	1.675	334.167	0.60	1.826
500	0.372	337.93	0.2512	1.8
1000	0.086	155.916	0.311	1.44

Table 4.2: The mean squared error for each model

In the following table, we can see how long each method takes based on the number of epochs.

Epochs	Approach 01	Approach 02	Second model	Low-rank approximation
50	39s	38s	37s	41s
100	1m 25s	2m 28s	2m 8s	1m 19s
500	7m 39s	7m 37s	6m 27s	6m 36s
1000	13m 37s	14m 31s	16m	12m 38s

Table 4.3: The timeliness for each model corresponding to the number of epochs

In the table 4.2, every model except the **approach 02**, has a low MSE, resulting in a good performance even-though the second approach has less parameters than the others, moreover it is time consuming. As we need to approximate a linear DtN map, it is not a useful approach for our problem, so fully connected layers are better.

To smooth the computational results, we use a digital filter, the so-called Savitzky–Golay filter. In general, such a filter is applied to a set of data points for the purpose of smoothing the data, i.e., increasing data precision without distorting the signal tendency. It uses the following polynomial form: $a_n x^n + \cdots + a_1 x + a_0$ to optimize the coefficients with at least $n + 1$ points.

We first apply it in case of the second model in the first approach. We show the smoothed approximation of the Neumann data using polynomial order 3. Also, the exact Neumann data is displayed in Figure 4.13.

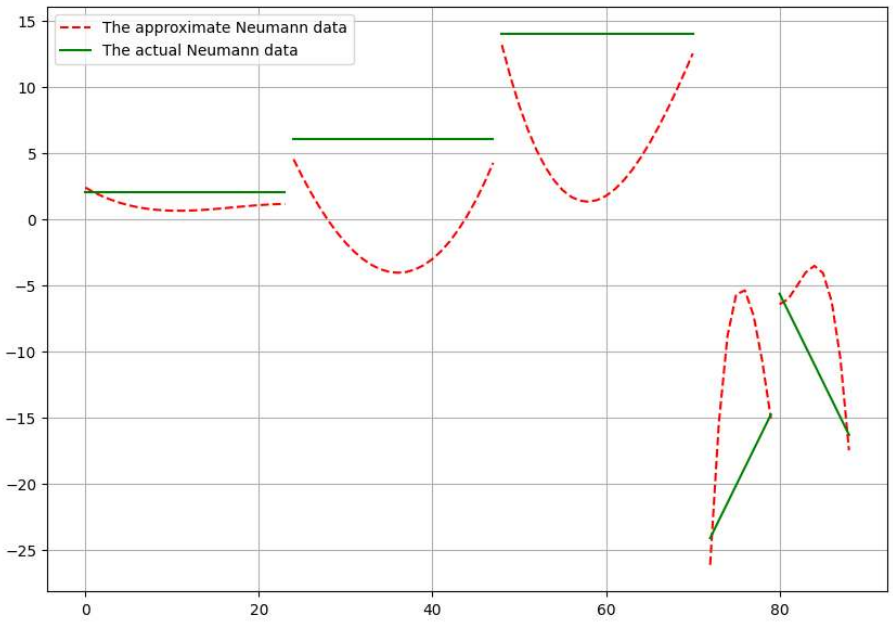


Figure 4.13: The exact and the approximated Neumann data in **approach 02**.

As soon as this model is eliminated, it remains to compare the other models. We can see in the following figures which model performs best based on the approximation of the DtN map, these results were obtained based on 1000 epochs.

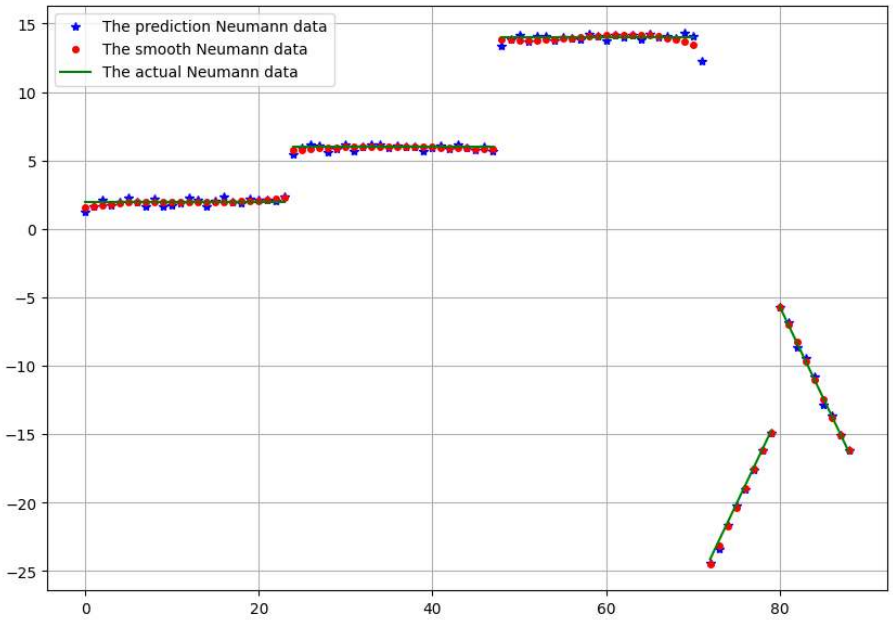


Figure 4.14: The real and the predicted Neumann data using **approach 01**.

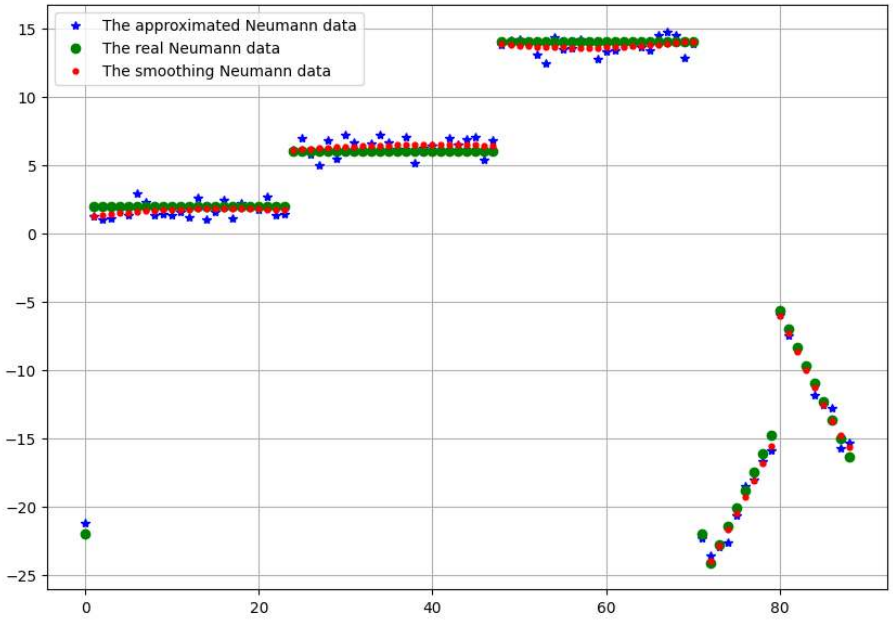


Figure 4.15: The real and predicted Neumann data using the **second approach**.

The actual and the estimation Neumann data using low-rank approximation method are shown in the following figure.

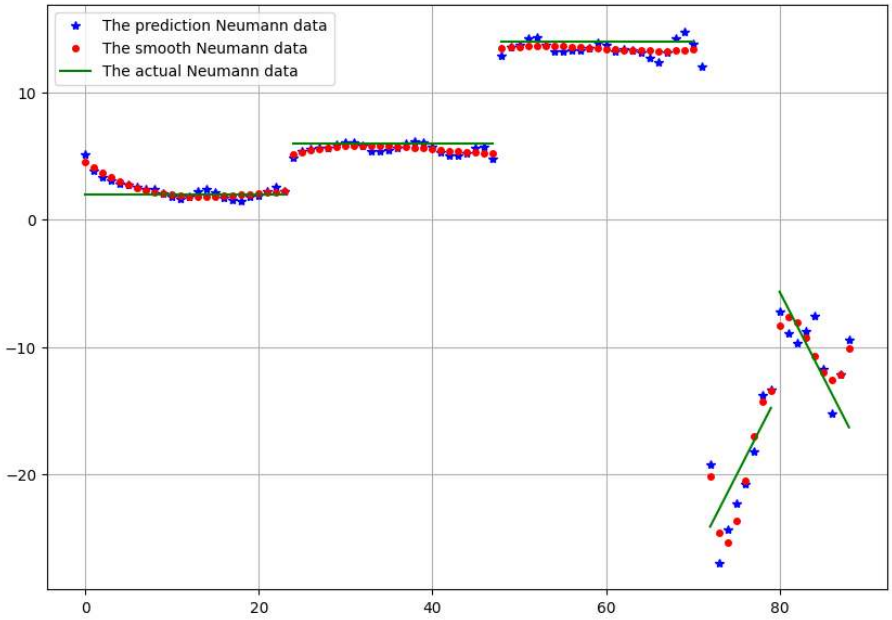


Figure 4.16: The real and predicted Neumann data using **low-rank approximation method**.

Chapter 5

Conclusion

We aim to find Dirichlet-to-Neumann maps and discuss their basic properties as well as their application through the conductivity equation in the electrical impedance tomography. We needed a related equation, which is the Shrödinger equation. Rather than using traditional approximation methods such as finite difference and finite elements methods, we used the most recent approximation method, provided by the neural networks. It is primarily based on optimization methods, learning from a given data and predicting results for unseen inputs.
According to our research, the best model for approximation of Laplacian problems is the first approach in which the neural network has only one dense layer. We also used another method and it gave a good result in terms of reducing the time complexity and parameters in order to use these parameters in the inverse problem, but it was less accurate compared to the first one.
The best method we chose uses only two layers, the input and output layers, and it is trained with the fundamental solution of the Laplace equation. This method worked as expected because of the linearity of the DtN map of the Laplace equation. The method is easy to implement, gives accurate results, and is not time-consuming, which is better than the classical methods, which suffer from high conditions numbers and highly depend on the discretizations. A deep learning model only needs a sufficiently wide data set to train so that it does not overfit or underfit and the model is ready for predictions.
A crucial step in using the method is choosing the neural network structure, such as the number of layers, the activation function, and the optimization method. As we need to deal with real data output, the DtN map of the Laplace problem is linear, so only one dense layer is required, and a linear activation function is the best option. To get the optimal value for the object function (weights), selecting the best optimization method is critical in terms of performance and time complexity. We tested a number of optimization methods in our research, but the Adam method performed the best. However, we also need to be careful when selecting the learning rate, because some problems do not require small learning steps, so the algorithm can reach the optimal value with just a few steps.
We can use this research as a study case for finding the DtN map of the Laplace problem in $2D$. The next step is to try similar models to approximate the DtN map in the 3-dimensional case. This research can also provide insights into

finding the DtN in homogeneous Laplace equations and applying deep learning methods to find the DtN map for non-homogeneous elliptic problems. This would be an important step towards the efficient approximation of inverse problems such as the electrical impedance tomography.

Bibliography

[1] Stochastic gradient descent. https://en.wikipedia.org/wiki/Stochastic_gradient_descent#Adam.

[2] Training, validation, and test data sets. https://en.wikipedia.org/wiki/Training,_validation,_and_test_data_sets, 2 August 2022,.

[3] Andy Adler, Romina Gaburro, and William Lionheart. Electrical impedance tomography. *Inverse Problems*, 2015.

[4] Akash Ajagekar. Adam. https://optimization.cbe.cornell.edu/index.php?title=Adam.

[5] Wolfgang Arendt, Ralph Chill, Christian Seifert, Hendrik Vogt, and Jürgen Voigt. Form methods for evolution equations, and applications. In *Lecture Notes of the 18th Internet Seminar on Evolution Equations (chapters 7 and 8)*, 2015.

[6] Jason Brownlee. Gentle introduction to the adam optimization algorithm for deep learning. https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/.

[7] Jason Brownlee. Train test validation split. https://www.v7labs.com/blog/train-validation-test-set.

[8] Luis Caffarelli and Luis Silvestre. An extension problem related to the fractional laplacian. *Communications in partial differential equations*, 32(8):1245–1260, 2007.

[9] Julianna Delua. Supervised vs unsupervised learning: What's the difference? https://www.ibm.com/cloud/blog/supervised-vs-unsupervised-learning, 2021.

[10] Lawrence C. Evans. *Partial Differential Equations*, volume 19 of *Graduate Studies in Mathematics*. American Mathematical Society, Providence, RI, 1998.

[11] Daniel Johnson. Supervised vs unsupervised learning: Difference between them. https://www.guru99.com/supervised-vs-unsupervised-learning.html. Accessed: 2022-08-30.

[12] Kuo-Ming Lee. A Source Problem for the Helmholtz Equation via a Dirichlet-to-Neumann Map. *Abstr. Appl. Anal.*, pages Art. ID 2298192, 7, 2021.

[13] William McLean and William Charles Hector McLean. *Strongly Elliptic Systems and Boundary Integral Equations*. Cambridge University Press, 2000.

[14] M. De Buhan P. Frey. *The numerical simulation of complex PDE problems, Chapter 2*. Universidad de Chile, Facultad de Ciencias Fisicas y Matematicas, 2008.

[15] peltarion.com. One dimension convolution layer. https://peltarion.com/knowledge-center/modeling-view/build-an-ai-model/blocks/1d-convolution, 2022,. [Online; accessed 2022].

[16] Bastian E. Rapp. Delta function. https://www.sciencedirect.com/topics/engineering/delta-function, 2013.

[17] John Sylvester and Gunther Uhlmann. The Dirichlet to Neumann map and applications. *Inverse problems in partial differential equations*, 42:101, 1990.

[18] Roger Temam. *Navier-Stokes Equations: Theory and Numerical Analysis*, volume 343. American Mathematical Soc., 2001.

[19] Eric W Weisstein. Green's function. *https://mathworld. wolfram. com/*, 2004.

[20] Sijue Wu. Green's function for the Laplacian. http://www.math.lsa.umich.edu/~sijue/greensfunction.pdf. Lecture notes.