UCL

4TH YEAR PROJECT

MATH0082

Machine learning for inverse scattering problems

Student
James Barker

Supervisor
Timo Betcke

2019/2020

Contents

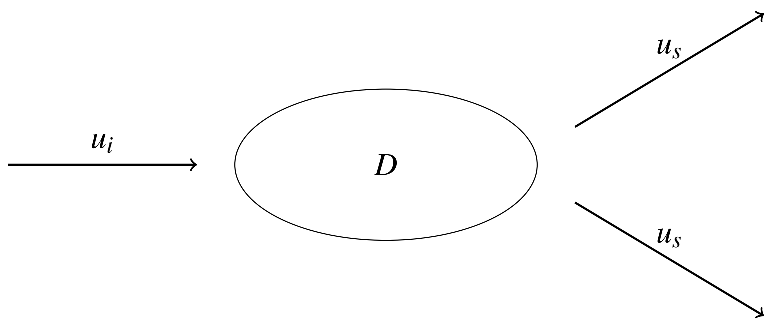

Figure 1.1: Inverse Scattering Illustration

1 Introduction

The definition of an inverse problem comes from a mapping from a set of parameters to data related to said parameters. The mapping is known as the forward operator and is what is trying to be inverted in an inverse problem. The forward problem is almost always well-posed, however the same cannot be said about the inverse problem. In this paper, we attempt to solve the acoustic inverse scattering problem.

The direct scattering problem involves the scattering of time-harmonic acoustic waves defined by

$$u(x,t) = e^{ikx\cdot d-\omega t},$$

through a homogeneous medium by an impenetrable bounded obstacle, D where $k = \frac{\omega}{c_0}$ is the wave number, ω is the frequency, c_0 is the speed of sound and d is the direction of propagation. The simple scattering problem is to find the total field u, s.t.

$$\triangle u + k^2 u = 0, \text{ in } \mathbb{R}^3, \tag{1.1}$$

$$u(x) = e^{ikx\cdot d} + u^s(x), \tag{1.2}$$

$$u = 0 \text{ on } \partial D, \tag{1.3}$$

$$\lim_{r\to\infty} r\left(\frac{\partial u^s}{\partial r} - iku^s\right) = 0, \tag{1.4}$$

Here the Helmholtz equation (1.1) is introduced along with the boundary condition of a sound-soft obstacle (1.3). In this paper, the far-field pattern is the data being used to reconstruct the original scattering parameters. In the theoretical section of this paper, Green's representation formula is introduced and used to derive a boundary integral equation. Consequently, the existence of a far-field map is introduced before exploring some of the properties of the far-field mapping.

In this paper, we focus purely on solving the inverse scattering problem by reconstructing the scattering obstacle from the far field pattern. The far field pattern is the solution of the direct scattering problem at a point where the distance from the scattering obstacle is much greater than that of the wavelength of the acoustic wave in the homogeneous medium. We chose the far-field pattern because it has more relevant applications, and since the 1980s many more mathematical properties surrounding it have been developed. In particular, considering the far field as a pattern of $\hat{x}$ in $L^2(\mathbb{S}^2)$, where $L^2(\mathbb{S}^2)$ is the space of square integrable patterns on the unit sphere $\mathbb{S}^2$ has allowed for development of theory concerning the completeness. However, the inverse of the far-field mapping from the scattered wave is ill-posed in the sense that the solution does not depend on the data and for many far field patterns a solution may not exist. Therefore, when using mathematical techniques to solve the inverse scattering problem, much care must be taken to avoid the blowing up of a non-existant solution.

The inverse scattering problem has many applications such as medical imaging, radar, sonar and geophysical exploration. However, most of these applications work with noisy real world data. On the other hand, the data used in this paper is very clean as it is generated using the Bempp-Cl [14] library in Python. If the results from this paper wish to be used in a more applied sense, it could be recommended to use the trained weights of one of the introduced models as initialised weights before training on the limited, noisy, real-world data. This technique is known as transfer learning and is very useful for machine learning tasks for which there is very limited data but a similar problem has an abundance of good quality data. Figure 5.1 gives examples of scattered patterns and the respective scattering obstacle, which were drawn using the Plotly library due to its strong capabilities of plotting 3D shapes.

Typically, inverse source problems are solved using a regularized Newton Method involving some form of Tikhonov Regularization, which is required due to the inverse problem being particularly ill-posed. Analytical solutions to inverse source problems are very rare and very often do not exist. Instead, numerical solutions must be found - which involve reformulating the problem using Green's Formulas and boundary integral equations before again reformulating the problem as a non-linear optimization problem. This technique requires a solution of the direct scattering problem at each iteration - which is very computationally expensive. However, these decomposition methods use two steps, the first constructs the scattered wave from the far field pattern - which overcomes the ill-posedness of the problem. The second step recovers the boundary by solving for where the boundary condition is satisfied in the least squares sense to overcome the non-linearity.

As previously discussed, inverse problems are trying to reconstruct parameters from data. The problem can be reformulated as learning the best estimator $\hat{y}$ for the operator y which maps the scattered far-field data to the parameters representing the boundary. This is the ideal task for machine learning, particularly as the majority of data in this context is computer generated and hence has two ideal properties in that it is clean and also is of an (effectively) unlimited supply. A lack of good data is one of the most common hindrances to most machine learning tasks, and overcoming this hurdle instantly is a strong incentive to attempt a machine learning solution to the inverse source problem. In this paper, it will be investigated whether a trained machine learning model is capable of quickly solving the inverse source problem while maintaining sufficient accuracy.

Machine learning has hit a boom in the last few years and shown very impressive performance on a range of complicated tasks. Deep learning in particular has achieved even superhuman performance on recent tasks thanks to modern research and a vast increase in computation power. It is also able to capture non-linear functions by using non-linear functions within the architecture of the model. An introduction to the theory behind some standard machine learning techniques is included in this text as well as some deep learning theory (related to standard feed forward networks).

After going through some theory of scattering and machine learning, we begin presenting results of machine learning on increasingly difficult tasks related to the inverse source problem. Initially we started with regressing the three radii that define an ellipsoid. We then tested out the ability of reducing the amount of data to the model along with its ability to generalise over big gaps of data. Then we moved on to the concept of voxels to allow the model to predict a more complex range of shapes before finally introducing multiple scattering shapes. Throughout, we compared a neural network's performance compared to some fundamental benchmarks which are introduced in the theoretical machine learning section. The results of the machine learning models are used to demonstrate the capability machine learning has to solve the inverse source problem rather than the finished state of the art (SOTA) model.

2 Theoretical Background

In this section, a solid theoretical background of the inverse source problem will be covered. Although we are studying an inverse problem, it is important to understand the theory of the corresponding direct problem. For this task, only acoustic waves have been focused on but a possible extension of this task would be to start working with electromagnetic waves as well. We are dealing in the domain of an isotropic, homogeneous medium (that we can treat as an inviscid fluid) in $\mathbb{R}^3$ that a sound wave is propagating through. We will then use Green's Equations twice. First to derive the integral boundary equations, then to derive the far field mapping before investigating some key properties of the far-field mapping that the inverse we are attempting to approximate.

2.1 Helmholtz Equation

2.1.1 Acoustic Waves

We denote the following:

- $v = v(x,t)$ be the velocity field.
- $p = p(x,t)$ be the pressure of the fluid (medium).
- $\rho = \rho(x,t)$ be the density.
- $S = S(x,t)$ be the specific entropy of the fluid.

We then model in the medium through the following equations:
Euler's equation:

$$\frac{\partial \upsilon}{\partial t} + (\upsilon \cdot \nabla)\upsilon + \frac{1}{\rho}\nabla p = 0 \tag{2.1}$$

Continuity equation:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \upsilon) = 0 \tag{2.2}$$

State equation:

$$p = f(\rho, S) \tag{2.3}$$

where f is a function depending on the state of the fluid. Adiabatic Hypothesis:

$$\frac{\partial S}{\partial t} = \upsilon \cdot \nabla S = 0 \tag{2.4}$$

The next step is to linearise the equations, by assuming that υ, p, ρ and S are small perturbations around:

- $\upsilon_0 = 0$.

- p_0, a constant,
- ρ_0, a constant,
- S_0, a constant,

respectively. These give the linearised equations:
Linearised Euler equation:

$$\frac{\partial \upsilon}{\partial t} + \frac{1}{\rho_0} = 0, \tag{2.5}$$

Linearised continuity equation:

$$\frac{\partial \rho}{\partial t} + \rho_0 \nabla \cdot v = 0, \tag{2.6}$$

Linearised state equation:

$$\frac{\partial p}{\partial t} = \frac{\partial f}{\partial \rho}(\rho_0, S_0)\frac{\partial \rho}{\partial t}, \tag{2.7}$$

From this we can define the speed of the acoustic wave $c = \sqrt{\frac{\partial f}{\partial \rho}(\rho_0, S_0)}$ and can from there form the wave equation:

$$\frac{1}{c^2}\frac{\partial^2 p}{\partial t^2} = \triangle p, \tag{2.8}$$

From the linearised Euler equation there exists a velocity potential $U = U(x,t)$ that has the following properties:

$$\upsilon = \frac{\nabla U}{\rho_0},$$

$$p = -\frac{\partial U}{\partial t}.$$

It is clear that the velocity potential will satisfy the wave equation:

$$\frac{1}{c^2}\frac{\partial^2 U}{\partial t^2} = \triangle U,$$

for time-harmonic acoustic waves of the form

$$U(x,t) = Re\left(u(x)e^{-i\omega t}\right),$$

where $\omega > 0$ is the wave frequency, we have that u satisfies the Helmholtz equation:

$$\triangle u + k^2 u = 0,$$

where $k = \frac{\omega}{c} > 0$ is the wave number.

We have an obstacle D with boundary δD. We also define the total wave u as $u = u^i + u^s$, where u^s denotes the scattered wave and u^i denotes the incident wave. We say an object is sound-soft if the total wave vanishes on the boundary i.e. we have the Dirichlet Boundary Condition $u = 0$ on δD. We would describe an object as sound-hard if we have perfect reflection of the wave on the boundary i.e. we have the Neumann Boundary Condition $\frac{\partial u}{\partial v} = 0$ on δD

where v is the outward normal to δD. For this project, we only investigate with sound-soft objects but it would be a possible, simple extension in future work to consider sound-hard objects.

However, as the density of the object D, ρ_D and speed of sound c_D are not equal to their corresponding values in the surrounding medium, a transmission problem arises (see [3]).

2.1.2 The Sommerfeld Radiation Condition

Arnold Sommerfeld ensured the uniqueness of exterior boundary value problems in 1912 by introducing his radiation condition [2]. Since then the condition has gone on to spawn much mathematical research in the uniqueness theorems of such exterior boundary value problems. It is a condition based at infinity and when applied to boundary value problems it isolates the solution to represent just the outgoing waves.

For the scattered wave u_s, the Sommerfeld Radiation Condition for $\mathbb{R}^3$ states:

$$\lim_{r\to\infty} r\left(\frac{\partial u^s}{\partial r} - iku^s\right) = 0,\ r = |x|. \tag{2.9}$$

This reduces the problem from the only two spherically symmetric solutions (obviously as the limit is taken independent of the two angular variables in cylindrical co-ordinates) of $\frac{e^{ix}}{|x|}$ and $\frac{e^{-ix}}{|x|}$ to just $\frac{e^{ix}}{|x|}$. This is because the other solution is unphysical as it can be interpreted as energy coming from infinity and sinking at zero. Furthermore, we have:

$$\text{Re}\left(\frac{e^{ik|x|} - i\omega t}{|x|}\right) = \frac{\cos k|x| - \omega t}{|x|},$$

corresponding physically to an outgoing wave.

We call a solution u *radiating* if u is a solution to the Helmholtz equation and it satisfies the Sommerfeld radiation condition.

2.2 Introducing Green's Representation Formula

One of the most fundamental tools used when studying the Helmoltz equations is Green's Integral Theorem.

Theorem 1 (Green's First Identity). *Let u and υ be scalar functions defined on some bounded domain D of class C^1. Let u be twice continuously differentiable and υ once differentiable. Let v be the outward pointing unit normal of surface element δS. Then*

$$\int_D (u \triangle \upsilon + \nabla u \cdot \nabla \upsilon)\, dx = \int_{\delta D} u \frac{\partial \upsilon}{\partial v}. \tag{2.10}$$

By interchanging υ and u, and subtracting the two, we have:

Theorem 2 (Green's Second Identity). *Let u and υ be scalar functions defined on some bounded domain D of class C^1. Let u and υ both be twice continuously differentiable. Let v be the outward pointing unit normal of surface element δS. Then*

$$\int_D (u \triangle \upsilon - \upsilon \triangle u)\, dx = \int_{\delta D} \left(u\frac{\partial \upsilon}{\partial v} - \upsilon \frac{\partial u}{\partial v} \right) \tag{2.11}$$

We also have the fundamental solution to the Helmholtz equation, which can be easily derived using Fourier transforms

$$\Phi(x,y) := \frac{e^{ik|x-y|}}{4\pi|x-y|},\ x \neq y \tag{2.12}$$

for fixed y and $x \in \mathbb{R}^3 \setminus \{y\}$:

Now we have D as a bounded domain of class C^2, v as the unit normal vector to the boundary δD directed into the exterior of D. Assuming $u \in C^2(D) \cap C(\bar{D})$ is a function which possesses a normal derivative on the boundary in the sense that the limit

$$\frac{\partial u}{\partial v} = \lim_{h \to \infty} (v(x)) \cdot \nabla u(x + hv(x))$$

exists uniformly on δD. We circumscribe an arbitrary fixed point $x \in D$ with the sphere S,

$$S(x,\rho) = \{y \in \mathbb{R}^3,\ |x-y| = \rho\},$$

contained in D. Now applying Green's Second Identity (2) using u and Φ we get:

$$\int_{\delta D \cup S(x,\rho)} \left(\frac{\partial u}{\partial v}\Phi(x,y) - u(y)\frac{\delta \Phi}{\delta v(y)} \right) ds(y)$$
$$= \int_D (\Phi \triangle u - u \triangle \Phi) ds(y),$$

and as we know Φ satisfies the Helmholtz equation, we have:

$$\int_{\delta D \cup S(x,\rho)} \left(\frac{\partial u}{\partial v}\Phi(x,y) - u(y)\frac{\delta \Phi}{\delta v(y)} \right) ds(y) = \int_D \Phi(\triangle u + k^2 u) ds(y). \tag{2.13}$$

On $S(x,\rho)$ we have $\Phi(x,y) = \frac{e^{ik\rho}}{4\pi\rho}$ and $\nabla_y \Phi(x,y) = \left(\frac{1}{\rho} - ik\right)\frac{e^{ik\rho}}{4\pi\rho} v(y)$. Subsequently, by applying the mean value theorem we have:

$$\lim_{\rho \to 0} \int_{S(x,\rho)} \left(\frac{\partial u}{\partial v}\Phi(x,y) - u(y)\frac{\delta \Phi}{\delta v(y)} \right) ds(y) = u(x). \tag{2.14}$$

Applying the same limit to (2.13) gives the following results

$$u(x) = \int_{\delta D} \left(\frac{\partial u}{\partial v}(y)\Phi(x,y) - \frac{\partial \Phi}{\partial v} u(x,y) \right) ds(y),\ x \in D. \tag{2.15}$$

This is known as Green's representation formula.

Similarly it can be shown that

$$u(x) = \int_{\partial D} \left(u(y) \frac{\partial \Phi(x,y)}{\partial \nu(y)} - \frac{\partial u}{\partial \nu}(y) \Phi(x,y) \right) ds(y),\ x \in \mathbb{R}^3 \setminus \bar{D} \tag{2.16}$$

These two variations 2.15 2.16 of Green's representation theorem can be seen to satisfy the sound-soft boundary condition.

2.3 Going from Green's Representation Formula to Boundary Integral Equations

2.3.1 Single and Double Layer Potential

We define the Acoustic Single Layer Potential as follows:

$$u(x) := \int_{\delta D} \varphi(y) \Phi(x,y) ds(y),\ x \in \mathbb{R}^3 \setminus \delta D \tag{2.17}$$

Since for $x \in \mathbb{R}^3 \setminus \delta D$, we can differentiate under the integral sign, we see that u is a solution to the Helmholtz equation.

We also have the Acoustic Double Layer Potential as:

$$\upsilon(x) := \int_{\delta D} \varphi(y) \frac{\partial \Phi(x,y)}{\partial \nu} ds(y),\ x \in \mathbb{R}^3 \setminus \delta D. \tag{2.18}$$

Which is also a solution to the Helmholtz equation with density φ. We have already shown that these are radiating solutions to Helmholtz equation in D and $\mathbb{R}^3 \setminus \bar{D}$. We also introduce the indices $+,-$ to denote the limits by approaching the boundary δD from inside $\mathbb{R}^3 \setminus \bar{D}$ and D respectively:

$$u_+(x) = \lim_{y \to x, y \in \mathbb{R}^3 \setminus \bar{D}} u(x),$$
$$u_-(x) = \lim_{y \to x, y \in D} u(x),$$

where $x \in \delta D$.

Additionally we have the following jump conditions across the boundary:

- $u_+ - u_- = 0$
- $\frac{\partial \upsilon_+}{\partial \nu} - \frac{\partial \upsilon_-}{\partial \nu} = \varphi$
- $\upsilon_+ - \upsilon_- = 0$
- $\frac{\partial u_+}{\partial \nu} - \frac{\partial u_-}{\partial \nu} = -\varphi$

Given an incident wave u_i that is a solution to the Helmholtz equation, we find a solution

$$u = u^i + u^s$$

to the Helmholtz equation in $\mathbb{R}^3 \setminus \bar{D}$ such that the following conditions are satisfied:

- $u = 0$ on δD
- u^s satisfies the Sommerfeld Radiation Condition.

This is a well-posed problem with an existing, unique solution but can be generalised to the Exterior Dirichlet Problem, where the boundary condition $u = 0$ on δD is replaced with $u = f$, where f is a continous function on δD. The Exterior Dirichlet Problem is also a well-posed problem with an existing, unique solution.

2.3.2 Indirect Method

First, we attempt to find a solution to the exterior Neumann problem in the form of a single-layer potential. To do this, rewrite u as a single layer potential as (2.17) with density φ to be found. For any reasonable φ, we have the Helmholtz equation and the Sommerfeld Radiation condition satisfied. Only the boundary condition of the exterior Neumann problem is left to fulfill. Using the jump condition for the single layer potential we obtain:

$$u(x) = \int_{\delta D} u(y)\frac{\partial \Phi}{\partial v} ds(y) = f(x),\ x \in \delta D, \tag{2.19}$$

If it is possible to solve this for u, the exterior Neumann problem is solved. However, for certain values of k it is impossible find a unique solution to this problem.

2.3.3 Direct Method

Instead, applying Green's theorem gives the integral representation for the exterior Neumann Problem.

$$2u(x) = \int_{\delta D} (\Phi(x,y)f(y) - u(y)\frac{\partial \Phi}{\partial v} ds(y),\ x \in D. \tag{2.20}$$

By letting x tend to the boundary δD we get:

$$u(x) + \int_{\delta D} u(y)\frac{\partial \Phi(x,y)}{\partial v} ds(y) = \int_{\delta D} \Phi(x,y)f(y),\ x \in \delta D. \tag{2.21}$$

Observe that (2.21) is the Hermitian Adjoint of (2.19) so it will have the same irregular values of k.

2.4 Showing the existence of a far field map

Solutions to the Helmholtz equation in all of $\mathbb{R}^3$ are called entire solutions. To satisfy the Sommerfeld radiation condition, from (2.16) and (2.13) it can be seen that the solution must vanish identically.

Theorem 3. *Every radiating solution u to the Helmholtz equation has the same asymptotic behaviour as an outgoing spherical wave*

$$u(x) = \frac{e^{ik|x|}}{|x|}\left(u_\infty(\hat{x}) + O\left(\frac{1}{|x|}\right)\right), |x| \to \infty$$

where

$$u_\infty(\hat{x}) = \frac{1}{4\pi}\int_{\delta D}\left(u(y)\frac{\partial e^{-ik\hat{x}\cdot y}}{\partial \nu} - \frac{\partial u}{\partial \nu}e^{-ik\hat{x}\cdot y}\right)ds(y), \hat{x}\in\mathbb{S}^2.$$

We can see this as $|x|\to\infty$ we have

$$|x-y| = \sqrt{|x|^2 - 2x\cdot y + |y|} = |x| - \hat{x}\cdot y + O\left(\frac{1}{x}\right).$$

We rewrite the fundamental solution Φ as

$$\Phi(x,y) = \frac{e^{ik|x|}}{|x|}\left(e^{-ik\hat{x}\cdot y} + O\left(\frac{1}{|x|}\right)\right), \tag{2.22}$$

and its derivative with respect to the outward unit normal as

$$\frac{\partial\Phi(x,y)}{\partial\nu(y)} = \frac{e^{ik|x|}}{|x|}\left(\frac{\partial e^{-ik\hat{x}\cdot y}}{\partial\nu(y)} + O\left(\frac{1}{|x|}\right)\right), \tag{2.23}$$

for all $y\in\delta D$. We then insert this into (2.16) which gives

$$u(x) = \frac{e^{ik|x|}}{|x|}\left\{u_\infty(\hat{x}) + O\left(\frac{1}{|x|}\right)\right\}, |x|\to\infty, \tag{2.24}$$

where the far field pattern is defined as follows:

$$u_\infty(\hat{x}) = \frac{1}{4\pi}\int_{\partial D}\left\{u(y)\frac{\partial e^{-ik\hat{x}\cdot y}}{\partial\nu(y)} - \frac{\partial u}{\partial\nu}(y)e^{-ik\hat{x}\cdot y}\right\}ds(y), \hat{x}\in\mathbb{S}^2. \tag{2.25}$$

So we now have an equation for the far-field, but will eventually be solving the equation $Au = u_\infty$, where A is the far-field operator. There are some key characteristics of this operator such as the ill-posedness of the inverse.

2.5 Characteristics of Far Field Map

In this section, some of the key concepts of the far-field mapping, A s.t. $Au = u_\infty$ will be covered. In particular, covering the injectivity and the ill-posedness. Initially, we show the the mapping is analytical.

2.5.1 Analytic on the Unit Sphere

It follows very clearly from (2.25) that the far-field mapping is analytical as it can be given as a convergent power series.

2.5.2 Injectivity from Radiating Solutions to Far Field

Rellich's Lemma will be used to prove the injectivity of the far field mapping. However, first some general background of spherical harmonics will be introduced by attempting to solve the Helmholtz equation in spherical co-ordinates. The Laplacian in spherical co-ordinates is:

$$\triangle = \frac{1}{r^2}\frac{\partial}{\partial r}\left(r^2\frac{\partial}{\partial r}\right) + \frac{1}{r^2\sin^2\phi}\frac{\partial^2}{\partial\theta^2} + \frac{1}{r^2\sin\phi}\frac{\partial}{\partial\phi}\left(\sin\phi\frac{\partial}{\partial\phi}\right).$$

To hope to solve the Helmholtz equation, we attempt the separation of variables technique:

$$u(r,\theta,\phi) = R(r)\Theta(\theta)\Phi(\phi)$$

So the Helmholtz equation in spherical co-ordinates becomes:

$$\left(\frac{r^2\sin^2\phi}{R}\frac{d^2R}{dr^2} + \frac{2r\sin^2\phi}{R}\frac{dR}{dr}\right) + \left(\frac{1}{\Theta}\frac{d^2\Theta}{d\theta^2}\right) + \left(\frac{\sin^2\phi}{\Phi}\frac{d^2\Phi}{d\phi} + \frac{\cos\phi\sin\phi}{\Phi}\frac{d\Phi}{d\phi}\right) = 0. \quad (2.26)$$

As the solution has rotational symmetry we only need to focus on the radial component of the solution which must equal a constant. So we have the spherical Bessel differential equation

$$r^2R'' + 2rR' + \left(r^2 - n(n+1)\right)R = 0, \quad (2.27)$$

which has two linearly independent solutions: j_n and y_n and have the following relation to the standard Bessel functions, J_n, Y_n:

$$j_n(x) = \sqrt{\frac{\pi}{2x}}J_{n+\frac{1}{2}}, \quad (2.28)$$

$$y_n(x) = \sqrt{\frac{\pi}{2x}}Y_{n+\frac{1}{2}}, \quad (2.29)$$

The linear combination denotes the spherical Hankel Function:

$$h_n^{1,2} := j_n \pm iy_n,$$

and now we have defined all the terms in the radiating solution to the Helmholtz equation.

The spherical Bessel and Neumann functions can also be expressed in terms of the following power series:

$$j_n(t) := \sum_{k=0}^{\infty}\frac{(-1)^k t^{n+2k}}{2^k k! 1\cdot 3\cdot\ldots\cdot(2k+1+2n)}, \quad (2.30)$$

and

$$y_n(t) := -\frac{(2n)!}{2^n n!}\sum_{k=0}^{\infty}\frac{(-1)^k t^{2k-n-1}}{2^k k!(1-2n)(3-2n)\ldots(2k-1-2n)}. \quad (2.31)$$

By equating powers of t from the power series representation of the spherical Bessel and Neumann functions that both satisfy the following recurrence relation:

$$f_{n+1}(t) + f_{n-1}(t) = \frac{2n+1}{t}f_n(t),\ n\in\mathbb{N}. \quad (2.32)$$

Similarly, by taking the derivative of the power series representation of the spherical Bessel and Neumann functions, the following differential recurrence relations are satisfied:

$$f_{n+1}(t) = -t^n \frac{d}{dt}\left(t^{-n} f_n(t)\right),\ n \in \mathbb{N}_0, \tag{2.33}$$

$$t^{n+1} f_{n-1}(t) = \frac{d}{dt}\left(t^{n+1} f_n(t)\right),\ n \in \mathbb{N}, \tag{2.34}$$

Then by taking the Wronskian, $W(f,g) = fg' - f'g$, we can see:

$$W' + \frac{2}{t}W = 0,$$

which obviously has the general solution

$$W(j_n(t), y_n(t)) = \frac{C}{t^2},$$

By taking the limit $t \to \infty$ the arbitrary constant C can be ignored. which gives:

$$j_n(t)y_n'(t) - j_n'(t)y_n(t) = \frac{1}{t^2}, \tag{2.35}$$

which gives the following result from the power series representation of the spherical Bessel function:

$$\lim_{n\to\infty}(j_n) = \frac{t^n}{1\cdot 3\cdot \ldots \cdot (2n+1)}\left(1 + O\left(\frac{1}{n}\right)\right). \tag{2.36}$$

Since we are in a on compact subset of $\mathbb{R}$ we have uniform converges. This leads to the limit for the spherical Hankel equation

$$\lim_{n\to\infty}(h_n) = \frac{1\cdot 3\cdot \ldots \cdot (2n-1)}{it^{n+1}}\left(1 + O\left(\frac{1}{n}\right)\right), \tag{2.37}$$

Then we have Stirling's Formula $n! \sim \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$, as $n \to \infty$. This gives the following spherical Hankel function's asymptotic behaviour:

$$h_n^1(t) = O\left(\frac{2n}{et}\right)^n,\ n \to \infty. \tag{2.38}$$

Combining (2.38) and (2.33) gives the two formulas:

$$h_n^{(1)}(t) = (-i)^n \frac{e^{it}}{it}\left\{1 + \sum_{p=1}^{n} \frac{a_{pn}}{t^p}\right\},$$

and

$$h_n^{(2)}(t) = i^n \frac{e^{-it}}{-it}\left\{1 + \sum_{p=1}^{n} \frac{\bar{a}_{pn}}{t^p}\right\},$$

which gives the following asymptotic behaviour

$$h_n^{1,2}(t) = \frac{1}{t} e^{\pm i\left(t - \frac{n\pi}{2} - \frac{\pi}{2}\right)}\left(1 + O\left(\frac{1}{t}\right)\right),\ \text{as } t \to \infty. \tag{2.39}$$

We are now in a position to prove Rellich's Lemma.

Theorem 4 (Rellich's Lemma)**.** *Assume D to be an open complement of an unbounded domain. Let $u \in C^2(\mathbb{R}^3)$ that satisfies*

$$\lim_{r\to\infty}\int_{|x|=r}|u(x)|^2 ds = 0,$$

then $u = 0$ in $\mathbb{R}^3 \setminus \bar{D}$

Proof: For large $|x|$ we can express $u(x)$ as:

$$u(x) = \sum_{n=0}^{\infty}\sum_{m=-n}^{n} a_n^m(|x|)Y_n^m(\hat{x}),$$

where Y_n^m denotes the spherical harmonics. It is possible to express $u(x)$ as the spherical harmonics as the spherical harmonics form a complete orthonormal system in $L^2(\mathbb{S}^2)$. Also, $\hat{x} = \frac{x}{|x|}$. The Fourier expansion coefficients are given by

$$a_m^n = \int_{S^2} u(r\hat{x})\bar{Y_n^m}(\hat{x})ds(\hat{x}),$$

and also satisfy Parseval's equality

$$\int_{|x|=r}|u(x)|^2 ds = r^2\sum_{n=0}^{\infty}\sum_{m=-n}^{n}|a_n^m(r)|^2 .$$

However, the assumption

$$\lim_{r\to\infty}\int_{|x|=r}|u(x)|^2 ds = 0,$$

obviously implies we have

$$\lim_{r\to\infty} r^2\,|a_n^m(r)|^2 = 0, \tag{2.40}$$

It can also be shown that the coefficients of the fourier expansion satisfy the spherical Bessel functions

$$\frac{d^2a_n^m}{dr^2} + \frac{2}{r}\frac{da_n^m}{dr} + \left(k^2 - \frac{n(n+1)}{r^2}\right)a_n^m = 0,$$

which gives the solution

$$a_n^m(r) = p_n^m h_n^{(1)}(kr) + q_n^m h_n^{(2)}(kr).$$

By combining this with (2.40) and the asymptotic behaviour of the spherical Hankel function (2.39), it is easily seen that the constants $p_n^m = q_n^m = 0$, $\forall m, n$, which gives the result $u = 0$. ■

Hence, we have established the injectivity of the mapping of radiating waves to the far-field pattern.

2.5.3 Ill-posedness of the the Inverse Mapping

In 1902 Hamard introduced the three conditions [1] for a well-posed problem:

1. A solution exists,

2. The solution is unique,

3. The solution continuously depends on the data

We will show the inverse mapping of the far-field mapping is ill-posed by demonstrating it contradicts the first and third criteria of Hamard's properties for a well-posed problem.

Firstly, we can express the radiating solution to the Helmholtz equation as

$$u(x) = \sum_{n=0}^{\infty} \sum_{m=-n}^{n} a_n^m h_n^1(k|x|) Y_n^m \left(\frac{x}{|x|} \right), \tag{2.41}$$

where the spherical harmonics $Y_n^m(\theta, \varphi) = \sqrt{\frac{2n+1}{4\pi} \frac{(n-|m|)!}{(n+|m|)!}} P_n^{|m|}(\cos\theta) e^{im\varphi}$. Also, h_n^1, denotes the spherical Hankel function which is the sum of a real component of the spherical Bessel function and an imaginary component of the spherical Neumann function. Both the spherical Bessel and Neumann functions are introduced when solving the Helmholtz equation in spherical co-ordinates.

We expand u in a basis of spherical harmonics:

$$u_\infty = \sum_{n=0}^{\infty} \sum_{m=-n}^{n} b_n^m Y_n^m$$

where $b_n^m = \int_{\mathbb{S}^2} u_\infty(\hat{x}) \overline{Y_n^m} ds(\hat{x})$ which is obviously going to be difficult to evaluate.

However, we do have the coefficients of the radiating solution a_n^m given by

$$a_n^m(r) = \int_{\mathbb{S}^2} u(r\hat{x}) \overline{Y_n^m}(\hat{x}) ds(\hat{x}),$$

We have then:

$$\begin{aligned} b_n^m &= \lim_{r \to \infty} \int_{\mathbb{S}^2} u_\infty(\hat{x}) \overline{Y_n^m}, \\ &= \lim_{r \to \infty} \int_{\mathbb{S}^2} re^{-ikr} u(r\hat{x})(\hat{x}) \overline{Y_n^m}, \\ &= \lim_{r \to \infty} re^{-ikr} \int_{\mathbb{S}^2} u(r\hat{x})(\hat{x}) \overline{Y_n^m}, \\ &= \frac{a_n^m}{ki^{n+1}}. \end{aligned}$$

So we have that the far field pattern of the radiating solution to the Helmholtz equation of the form given above is:

$$u_\infty = \frac{1}{k} \sum_{n=0}^{\infty} \frac{1}{i^{(n+1)}} \sum_{m=-n}^{n} a_n^m Y_n^m$$

Parseval's Equation for (2.41) becomes:

$$r^2 \sum_{n=0}^{\infty} \sum_{m=-n}^{n} |a_n^m|^2 |h_n^1(kr)|^2 = \int_{|x|=r} |u(x)|^2 ds(x)$$

Combining this with the asymptotic behaviour of the spherical Hankel function (2.38) for large n, we have the growth condition for the coefficients as:

$$\sum_{n=0}^{\infty}(\frac{2n}{ker})^{2n}\sum_{m=-n}^{n}|a_n^m|^2 \leq \infty. \tag{2.42}$$

We can treat the far-field pattern as a linear operator, A, that maps a radiating solution u to the Helmholtz equation onto its far field equation, u_∞. Then we can solve

$$Au = u_\infty \tag{2.43}$$

However, the growth condition (2.42) causes an issue when trying to solve the inverse problem (2.43) for the far field operator as there will be many u_∞ for which a solution u does not exist. Hence this contradicts one of Hamard's three conditions [1] for a well-posed problem:

- A solution exists,
- The solution exists,
- The solution continuously depends on the data

Evidently, the inverse far-field problem does not match these criteria and hence the problem is ill-posed. As well as the lack of existence of a solution in some cases, suppose a solution did exist, it would not continuously depend on u_∞ in any reasonable norm.

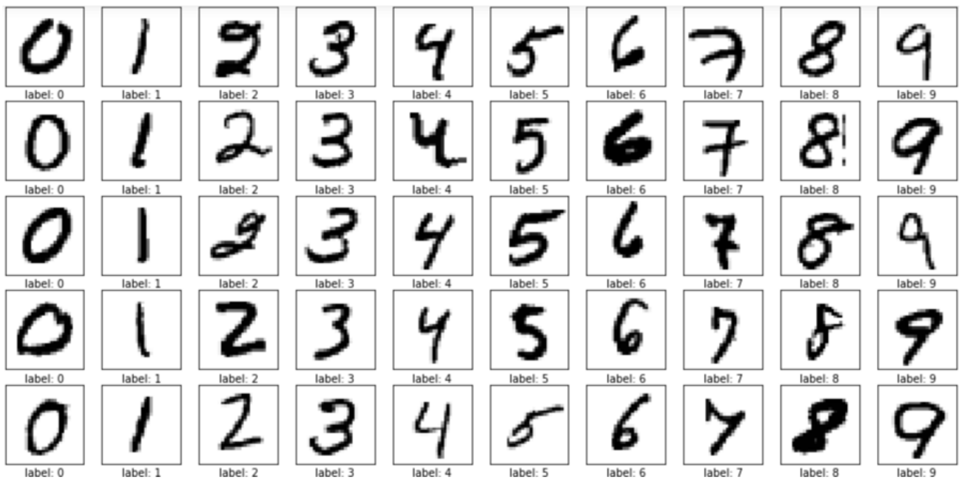

Figure 3.1: A Sample of Handwritten Digits from the MNIST [9] dataset

3 The Fundamental Concepts of Machine Learning

3.1 Using Machine Learning

Machine Learning is a term that was popularized by Arthur Samuel in 1959. He states machine learning is "The field of study that just gives computers the ability to learn without being explicitly programmed". For example, take the task of getting a computer to recognise a handwritten digit as seen in Figure 3.1. Each digit consists of $28^2 = 784$ pixels, of value 1 or 0. It could be approached naively by creating a large amount of rules to distinguish the shapes into each digit, but this would lead to a very complicated and long list of rules and probably an overall poor performance of the system due to the large discrepancies among different people's handwriting. Instead, you can vectorise the input pixels for each digit, $\boldsymbol{x}$, create a large set of these vectorised digits and use it to tune the parameters of an adaptive model that would map each vector to an integer, y representing the corresponding digit. The recent boom of GPU power has allowed for training of these adaptive models to perform well on a plethora of more complicated tasks. This hand-written digit recognising task is often referred to as the MNIST task [9] and is the "Hello World" equivalent of deep learning (a branch of machine learning).

Machine Learning can be divided into two areas, supervised and unsupervised. Supervised learning is where the problem consists around using labelled data, such as each image of a handwritten digit being labelled with that of its corresponding digit. Unsupervised learning is the task of discovering new patterns in an unlabelled dataset. An example of this applied to the MNIST task would be clustering a dataset of handwritten images into 10 different groups to show that there are 10 different digit types from the dataset. Supervised machine learning is more relevant in the problem that machine learning is going to be applied to in this paper, so some supervised learning techniques will be covered in more detail.

3.2 Supervised Machine Learning

Supervised machine learning can be divided into two different tasks: regression and classification. Regression is the task of predicting a continuous output such as house prices, while classification is the task of predicting a discrete output such as which digit a handwritten number is depicting. One of the most basic and common classifications model is that of Logistic Regression. As Logistic Regression was used as a baseline model (a model to compare the success of other models), a theoretical overview will be presented.

3.2.1 Logistic Regression

Logistic Regression is a probablistic model that takes the log-odds ratio

$$a = \ln \frac{P(y=1 \mid x)}{P(y=1 \mid x)},$$

models it as a linear function:

$$a \approx w_0 + w_1 x_1 + \ldots + w_m x_m,$$

and maps it to the logistic function:

$$\sigma = \frac{1}{1+e^{-a}},$$

It then has a linear decision boundary where $a = 0$ or where $\sigma = \frac{1}{2}$.

Logistic Regression learns it weights by attempting to maximise the likelihood, $L(D)$ defined by:

$$L(D) = P(y_1, y_2, \ldots y_n \mid x_1, x_2, \ldots x_m, \boldsymbol{w}) = \prod_{i=1}^{n} \sigma(\boldsymbol{w}^T \boldsymbol{x})_i^y \sigma(1 - \boldsymbol{w}^T \boldsymbol{x})^{1-y_i},$$

when y_i takes on values $0, 1$. However, likelihood is a difficult function to maximise as taking the product of a large amount of small numbers is computationally unstable so instead it attempts to maximise the log-likelihood:

$$l(D) = \log(L(D)) = \sum_{i=1}^{n} \big(y_i \log(\sigma(\boldsymbol{w}^T)) + (1-y_i)\log(1-\boldsymbol{w}^T)\big).$$

This maximisation task of $l(D)$, can be transformed into a more classical minimisation task by minimising $-l(D) = CE(D)$, which is often referred to as cross-entropy. This can be minimised using gradient descent by taking the gradient of cross entropy. First, we can calculate the gradient using the chain rule:

$$\frac{\partial CE}{\partial \boldsymbol{w}} = \frac{\partial CE}{\partial \sigma(\boldsymbol{w}^T \boldsymbol{x})} \frac{\partial \sigma(\boldsymbol{w}^T \boldsymbol{x})}{\partial \boldsymbol{w}^T \boldsymbol{x}} \frac{\partial \boldsymbol{w}^T \boldsymbol{x}}{\partial \boldsymbol{w}}$$

we have that

$$\frac{\partial \sigma(\boldsymbol{w}^T \boldsymbol{x})}{\partial \boldsymbol{w}^T \boldsymbol{x}} = \sigma(1-\sigma)$$

and

$$\frac{\partial 1 - \sigma(\boldsymbol{w}^T\boldsymbol{x})}{\partial \boldsymbol{w}^T\boldsymbol{x}} = -\sigma(1-\sigma).$$

So we have

$$\frac{\partial CE}{\partial \boldsymbol{w}} = \sum_{i=1}^{n} y_i \frac{1}{\sigma(\boldsymbol{w}^T\boldsymbol{x})}(1-\sigma(\boldsymbol{w}^T\boldsymbol{x}))\sigma(\boldsymbol{w}^T\boldsymbol{x})\boldsymbol{x} - (1-y_i)\frac{1}{1-\sigma(\boldsymbol{w}^T\boldsymbol{x})}(1-\sigma(\boldsymbol{w}^T\boldsymbol{x}))\sigma(\boldsymbol{w}^T\boldsymbol{x})\boldsymbol{x},$$

$$= \sum_{i=1}^{n} x(y_i(1-\sigma(\boldsymbol{w}^T\boldsymbol{x})) - (1-y_i)\sigma(\boldsymbol{w}^T\boldsymbol{x})),$$

$$= \sum_{i=1}^{n} x(y_i - \sigma(\boldsymbol{w}^T\boldsymbol{x})).$$

So we have the gradient descent scheme

$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k + \alpha_k \sum_{i=1}^{n} x(y_i - \sigma(\boldsymbol{w}^T\boldsymbol{x})),$$

for some learning rate $a_k > 0$.

Later on in this chapter, we go over what overfitting and underfitting is - but one of the large signs of overfitting is the model parameters taking on large values. In order to regularise this in the context of Logistic Regression, we introduce an L2 penalisation term in the weights. In this, the minimisation problem becomes

$$\min_{\boldsymbol{w}} CE(\boldsymbol{w}) + \lambda ||\boldsymbol{w}||_2,$$

for some regularisation constant, λ. Then the gradient descent scheme becomes:

$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k + \alpha_k \left(\sum_{i=1}^{n} x(y_i - \sigma(\boldsymbol{w}^T\boldsymbol{x})) + 2\lambda \boldsymbol{w}_k \right).$$

3.2.2 Decision Trees

Another, more complex benchmark used in the paper is that of a random forest [6]. Random forests are an ensemble method of decision trees. Hence, we shall cover a theoretical basis of decision trees first. Decision trees consist of:

- nodes representing a partition of the input space,
- internal nodes which are tests on values of different features,
- leaf nodes that include a subset of training examples that satisfy the tests along the branch. These ultimately determine the classification.

Decision trees are a popular machine learning model because of their ability to easily handle a mixture of continuous and discrete data and how easy it is to interpret the predictions.

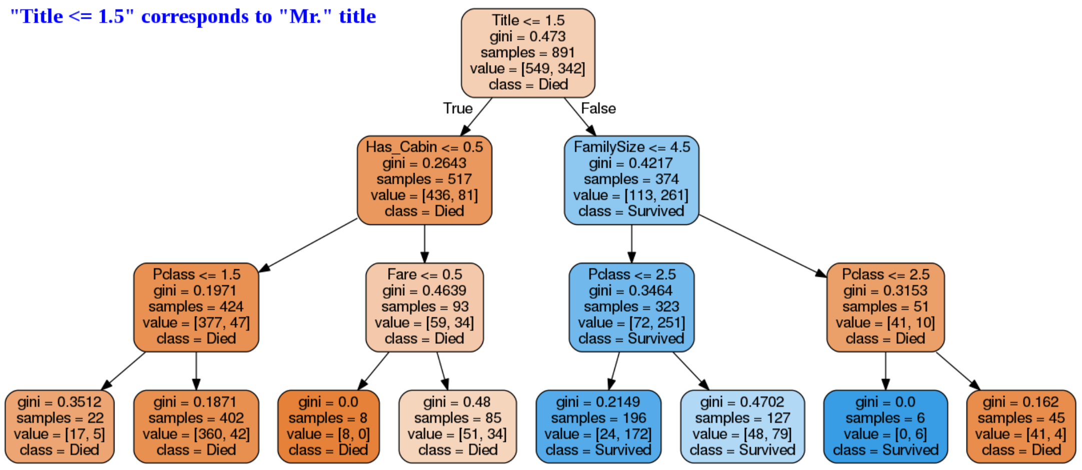

Figure 3.2: A decision tree trained on the predicting the survivors of the Titanic dataset [13]

The interpretability comes from the fact that a decision tree can always be converted into an equivalent set of if-then rules. A test on an internal node typically tests on one feature of the input space. A decision tree learns which test is best to ask by trying to maximise the information gain at each internal node, i.e. attaining the best split at each node. For example, in the game of "Guess Who?" where you have to ask questions on different characters appearance's to deduce which person your opponent has, it makes more sense to guess first the character's gender rather than whether they are wearing glasses. Information gain can also be interpreted as the change of entropy after a test. In this case the entropy is defined for a current state S,

$$E(S) = -\sum_{i=1}^{n} p_i \log p_i,$$

where p_i is the probability of the class i being in the state S. Another possible interpretation of Entropy is the Gini Index:

$$Gini(S) = 1 - \sum_{i=1}^{n} p_i^2.$$

The algorithm then goes through each possible split at each level to work out which test gives the largest information gain. Decision trees are very prone to overfitting, so quite often the max depth of a decision tree is restricted to prevent this or pruning techniques are used to limit the complexity of the models.

3.2.3 Random Forests

Random Forests are an example of an ensemble machine learning algorithm. Ensemble learning algorithms are meta-algorithms that combine several machine learning models in the hope of improving predictions. Ensemble methods have proven to be very powerful on a large range of machine learning tasks, including the Netflix Prize - an open competition for the best collaborative filtering algorithm to predict user ratings for films with a prize of $\$1,000,000$ [8]. When models overfit, there is a large variance and low bias - so a random forest tries to reduce the variance of these overfitting models using the Central Limit Theorem.

Theorem 5 (Central Limit Theorem)**.** *Let $X_1, X_2, ..., X_n$ be a set of N independent, identically distributed random variables with mean μ and finite variance σ^2. Then as $n \to \infty$:*

$$\lim_{n\to\infty} P\left(\mu + \frac{\sigma a}{\sqrt{n}} \leq \bar{X}_n \leq \mu + \frac{\sigma b}{\sqrt{n}}\right) = \Phi(b) - \Phi(a),$$

where $\bar{X}_n = \frac{1}{n}\sum_{i=1}^{n} X_i$ and $\Phi(x)$ denotes the distribution function of the standard normal distribution.

This can also be interpreted as $\bar{X}_n \sim N(\mu, \frac{\sigma}{\sqrt{n}})$, so for large n the variance is greatly reduced. The random forest uses a bagging ensemble technique that trains many decision trees on random subsets of the training dataset with replacement of the original size of the training dataset. Additionally, random forests implement random feature selection by selecting $k \ll m$ features to find the best test to separate the data, where m is the total number of features. The ensemble model then aggregates the output of all the models before predicting. This means random forests consist of decision trees trained on different sets of data and different features. One drawback of the random forest model from decision trees is that the interpretable aspect of the model is lost.

We have now covered a solid grounding of some classical machine learning techniques. Now we shall move on to deep learning - a subset of machine learning that use multiple layers of perceptrons. Deep learning models proved the most successful of the models used to attempt to solve the inverse scattering problem.

3.3 Deep Learning

The model mainly used in this paper was a shallow neural network. We shall cover mathematical concepts behind how a neural network predicts and is trained. First, we will cover a simple neuron.

3.3.1 Single Neuron

A neural network consists of layers of single neurons. A neuron takes in m inputs for one sample of the training set. The single neuron will then take a weighted sum of the inputs before passing it through a non-linear activation function.

$$z = \boldsymbol{w} \cdot \boldsymbol{x} + b = w_1 x_1 + w_2 x_2 + ... + w_n x_n + b,$$

the output y is then a non-linear function $g(z)$ where b is a learnt bias term and $\boldsymbol{w}$ are learnt weights.

3.3.2 Single Layer

We will now notate the ith layer in a neural network consisting of single neurons as a^i and a single neuron j in that layer by a^i_j. We then have the output nodes in the next layer a^{i+1}, denoted as follows:

$$a_{i+1,j} = g_{i+1}\left(\boldsymbol{w}_{i+1,j} \cdot \boldsymbol{a}_i + b_{i+1,j}\right),$$

or summarised even more effectively in matrix form:

$$a_{i+1} = g_{i+1}\left(\boldsymbol{W}_{i+1} \cdot \boldsymbol{a}_i + \boldsymbol{b}_{i+1}\right),$$

where

$$\boldsymbol{W}_i = \begin{bmatrix} \boldsymbol{w}_{i,1}^T \\ \boldsymbol{w}_{i,2}^T \\ \vdots \\ \boldsymbol{w}_{i,p}^T \end{bmatrix}$$

where p is the the number of neurons in the layer and $w_{i,j}$ represents the learned weights for the jth neuron in the ith layer. Similarly,

$$b_i = \begin{bmatrix} b_{i,1} \\ b_{i,2} \\ \vdots \\ b_{i,p} \end{bmatrix}$$

where $b_{i,j}$ is the learned bias term for each neuron.

We can then easily see how networks with multiple layers can be calculated.

3.3.3 Activation Functions

A neural network without activation functions would just be stacking many linear functions of each other hence causing the overall function to be linear. Therefore one key characteristic of activation functions is that they must be non-linear. Also, as the training of neural networks is dependent on the calculating of gradients, they must have a derivative that is easy to calculate (especially to speed up the training which can be costly).

Figure 3.3 shows some popular activation functions, which are defined as follows:

- sigmoid, $g(x) = \frac{1}{1+e^{-x}}$. This is popular for classification tasks as it can be interpreted as an output of a probability (bounded between 0 and 1). However, as it has a small range it is hard to train as for absolutely large inputs, the output is very similar. It was used as the output layer of the voxels task used in this paper (as that is predicting many binary classifications as a probablistic output).
- tanh, $g(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$. This has a very "nice" derivative of $1 - tanh^2(x)$. Similarly to the sigmoid function it can be interpreted as a probability and is less "saturated" than the sigmoid function (as it is bounded between -1 and 1 instead of 0 and 1) although it has a similar issue related to large inputs.
- ReLU, $g(x) = max(x, 0)$. The Rectified Linear Unit (ReLU) is the most common activation function chosen, having achieved very strong empirical results on a number of tasks. The derivative is very easy to calculate, and has an unbounded range. It is what was used in any hidden layers of the network in this paper.

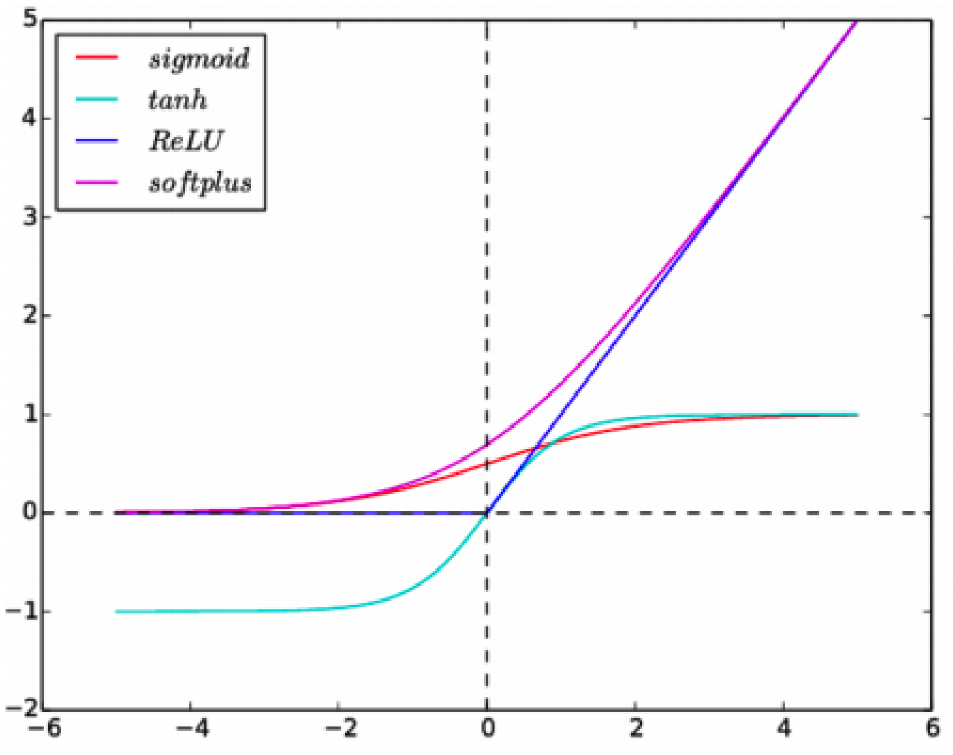

Figure 3.3: Different Activation Functions [15]

- softplus, $g(x) = \ln(1+e^x)$. Is a continuous variant of ReLu (among many over variants recently suggested). However, the bonus of it being continuous and appeasing the mathematician within us is negated by the fact that the derivative is much more expensive to calculate than ReLU.

With these non-linear functions, it allows the network to approximate very complicated non-linear functions (such as the inverse-scattering operator we hope). In fact, it has been shown [4] that a feed-forward network with a single hidden layer containing a finite number of neurons can approximate continuous functions in $\mathbb{R}^n$ if and only if the activation function is not a polynomial [5].

The choice of activation function is not an exact science like many areas of deep-learning. Many choices made are based upon previous empirical results rather than exact proofs, so most choices were made upon previous experience and observing techniques used in other papers.

3.3.4 Training of a Neural Network

Since we have covered how to evaluate a neural network, we will now consider how to learn the optimal weights for the task. We have already talked about how it is required that the activation functions are differentiable, this is because gradient descent is used to calculate optimal weights for the network. Gradient descent is an algorithm that requires the derivative of the function it is minimising. This is reformulating the problem to an optimisation task of finding the optimal network weights to minimise the loss. Therefore it is imperative to define the loss function that the network is trying to minimise with respect to the network parameters. In this paper, we have two type of tasks: a regression and a classification. For the regression problem, we define it to minimise the Mean Squared Error

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 .$$

For the classification task, we want to minimise the loss of the many binary classifications made by the model so define the loss as

$$H(p) = -p\log(1-p) - (1-p)\log(p).$$

Since each individual node is the output of a function that we know the derivative of, we can use the chain rule to calculate the error with respect to each parameter of the model. This is done by summing over all paths from each node to the output. However, this naive approach causes a combinatorial explosion as the number of paths between two nodes can be exponential in the number of graph edges and is very large in a fully connected feed-forward network. Instead, the paths can be factored by summing all incoming derivatives/edges at each node of the graph. As we want to calculate the derivative of the output with respect to all the previous layers, we start with the output of the network and work backwards. This allows the calculation of the derivative for all weights with respect to the error in one pass of the network. This technique of reversed automatic differentiation is called backpropagation.

Since it is now possible to explain how to calculate the derivatives, we can now use gradient descent to minimise the loss of the function. Gradient Descent can be explained very simply for a parameter w with the following algorithm:

$$w_{k+1} = w_k - \alpha_k \frac{\partial Err(w_k)}{\partial w_k},$$

where $\alpha_k > 0$, is the learning rate for iteration k and Err is our loss (error) function.

3.3.5 Optimisation

Neural Networks are trained on a large amount of data, so it would be incredibly expensive and slow to calculate the newly updated weights only after calculating the error for the entire dataset. Instead, the gradient descent is calculated on mini-batches of the training dataset, this is called Stochastic Gradient Descent and allows for quicker computation of an optimal set of weights. Convergence of Gradient Descent and Stochastic Gradient Descent is guaranteed to a local minima, however not a global minima. To try and escape local minima and find more optimal minima, different varations of gradient descent are used. In this paper, the optimiser frequently used was Adam [11], which uses the following algorithm:

Requiring the following:

- m_0, the first initial moment vector
- v_0, the second initial moment vector
- w_0, the initial parameter vector

- $J(\theta)$, the loss function of the network
- β_1, β_2 are the exponential decay rates for the momentum. Typically, $\beta_1 = 0.9$ and $\beta_2 = 0.999$
- ε, which is typically = 10^{-8}
- α, the learning rate

that are updated using the following algorithm:

$$t = t + 1,$$

$$\begin{aligned} g_t &= \nabla_w J(\theta_{t-1}), \\ m_t &= \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t, \\ v_t &= \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2, \\ \hat{m}_t &= \frac{m_t}{1 - \beta_1^t}, \\ \hat{v}_t &= \frac{v_t}{1 - \beta_2^t}, \\ w_t &= w_{t-1} - \alpha \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \varepsilon}. \end{aligned}$$

Adam combines two other optimisers which are using SGD with Momentum and RMSProp. It stores an exponentially decaying average of previous squared gradients in v_t but also an exponentially decaying gradient average of previous gradients which is similar to a momentum term. Both v_t and m_t are then bias-corrected before being using to adjust the learning rate in the update of weights. It has performed well empirically, particularly at speeding up the early stages of training (it has been found that Stochastic Gradient Descent performs better at finding smaller minima in later stages of training).

Another hyper-parameter available for tuning is the learning rate and batch size of the training. Batch size is the number of datapoints that will be propagated through the network before the weights are updated. Smaller batch sizes allow for quicker, more stochastic updates and usually a batch size of between 32 and 128 are used. Increasing batch size has been observed to have a similar effect to that of decreasing the learning rate. A large learning rate/small batch size will often lead to fast initial learning rate but struggle to find a lower minima and conversely a small learning rate/large batch size will lead to a lower minima but will require many more iterations.

A common technique used when training a machine learning model is to reduce the learning rate upon plateau of the training error [10]. This allows for faster learning in the earlier stages of training but after the error plateaus, the learning rate is reduced allowing it to find a more optimal minima towards the end of the training. The technique works more evidently on optimisrs that don't have adaptive learning rates (for example vanilla gradient descent). The reduction of learning rate technique is used very often and causes a characteristic elbow shape in the loss curves. These are demonstrated from the SOTA deep Residual Network paper [12] in the loss curves in 3.5.

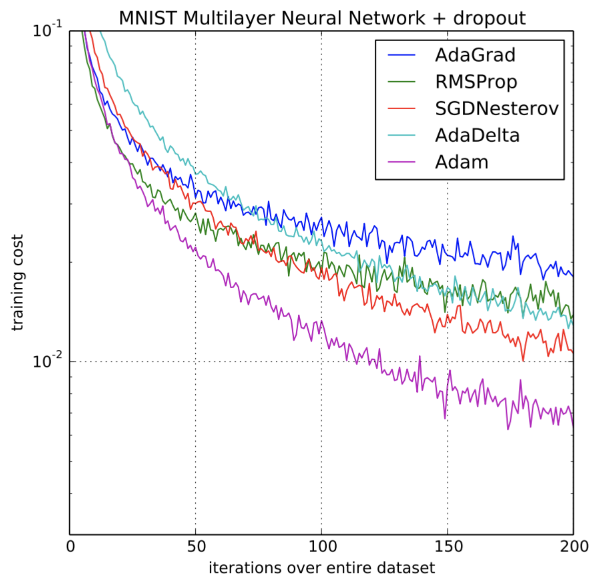

Figure 3.4: Comparison of Performance of Different Optimisers [11]

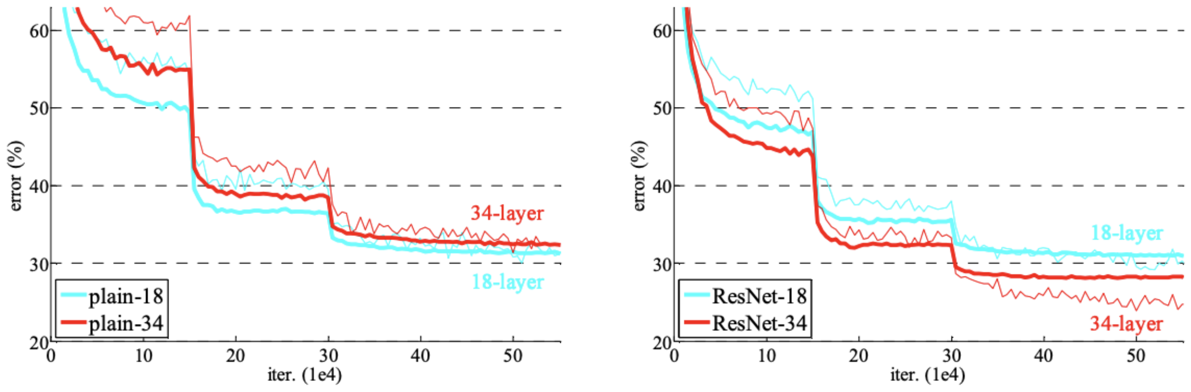

Figure 3.5: Loss Curves for deep Residual Networks on ImageNet [12]

Model	Train MSE	Validation MSE
Linear	0.369	0.340
Quadratic	0.153	0.150
Deg10 Polynomial	0.054	0.307

Table 3.1: Results of different degree polynomials fitting a noisy quadratic

3.4 Underfitting/Overfitting

As neural networks often have lots of parameters they are able to approximate very complex non-linear functions. However, this does have the drawback that they are very prone to overfitting, therefore it is important to monitor the networks training to ensure it is generalising well.

Overfitting is when a model fits the seen data too well causing it to fail to perform well on unseen/future data it may be tested on. Underfitting is when a model is inadequately capturing the seen data. An example to explain this is if you were trying to model a set of data coming from a quadratic with an added bit of noise: i.e $f(x) = ax^2 + bx + c + \varepsilon$ where epsilon is small random noise. If you tried to capture this data using a linear regression model, it would be unable to capture the complexity of the data. However, if you used a polynomial interpolation of degree 10 it would be able to capture well the seen data but would not generalise well to unseen data from the noisy quadratic. We can demonstrate this by looking at Figure 3.6, where we show the three previously described models trying to fit 14 points from the previously described noisy polynomial. These models are then evaluated on the seen 14 points (the training set) and 14 unseen points (validation set). We can observe that although the polynomial of degree 10 (Deg10 Polynomial) manages to get very close to the points in the training set, it doesn't get close to the points in the validation set. This is evident when calculating the MSE, shown in Table 3.1, where the training error is much lower than the validation error. This is the key sign of overfitting. Also, observe that the linear model captures neither the validation nor training set well, evidenced by the poor training and validation results. This is the key sign of underfitting. We see that the quadratic model, as we would expect, performs best on this task as it neither underfits nor overfits and has a low training error (although larger than the Deg10 Polynomial) but more importantly the lowest validation error.

By splitting data into training and validation sets, it allows the monitoring of how well a model is doing. Monitoring the training error gives indications on whether a model is underfitting. Similarly, monitoring the validation error gives indications on how well the model is generalising. The validation error is also used to tune hyper-parameters of the model, therefore it is important to separate another fraction of the data referred to as the test dataset. This is to ensure that the tuning of the hyper-parameters isn't leading to an overfitting on the validation set.

When training a neural network, initially the model is underfitting but as training continues, the model transitions goes from underfitting to overfitting. It is then important to find the sweet-spot between these two where the model is performing well on the training task and still generalising well to the validation set. Traditionally, in machine learning this is done by evaluating the model on the training and validation sets after a set number of batch updates.

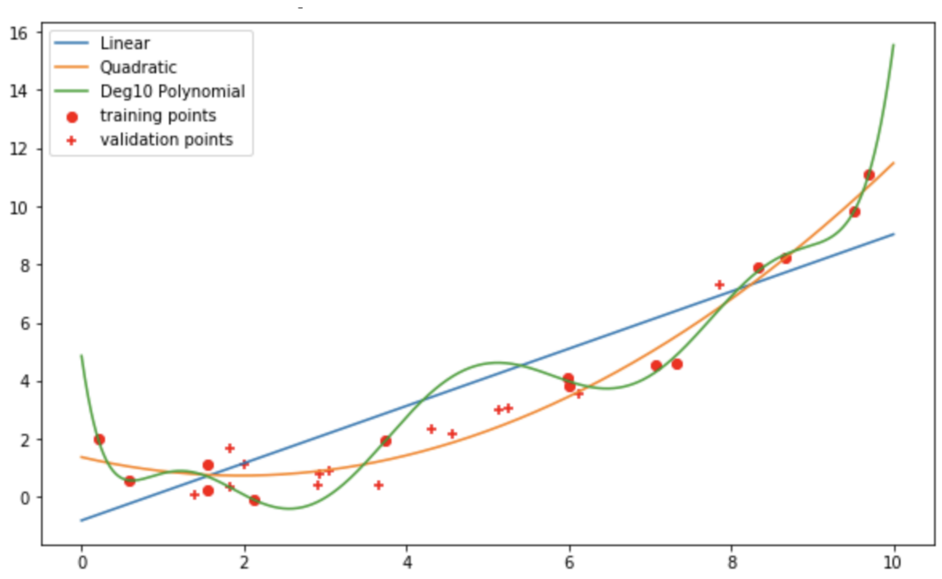

Figure 3.6: Comparing different degree polynomials fitting a noisy quadratic

This is usually one iteration over an entire dataset and called an epoch. To ensure a well generalising model, Early Stopping is used in training. Early Stopping is when the model stops training before converging in hope to prevent overfitting. It is implemented by monitoring the validation error and if it doesn't improve after a certain number of epochs, the training is stopped and the previous best weights are stored. Figure 3.7 is a typical loss curve for different epochs. Observe that the minimum of the validation loss curve is after the point where the training error separates from the validation error. This means that a small amount of overfitting (i.e. Training Error < Validation Error) leads to a better result.

Underfitting and overfitting can also be considered in terms of a tradeoff between bias and variance. Bias can be defined as

$$\mathbb{E}(\hat{y}(x)) - y(x),$$

and interpreted as how far off is our mean of prediction from the correct result. Variance is defined as

$$\mathbb{E}\left(\hat{y}(x)^2\right) - \mathbb{E}(\hat{y}(x))^2,$$

and can be interpreted as how wildy the predictions vary. An overly simple model, like the linear model will have a high bias and low variance whereas an overly complex model will have a low bias but high variance. Hence, it can be interpreted that overfitting corresponds to a low bias and high variance and underfitting vice versa. An ideal model has low bias and low variance and hence is why the ensemble method of random forests attempts to lower the variance.

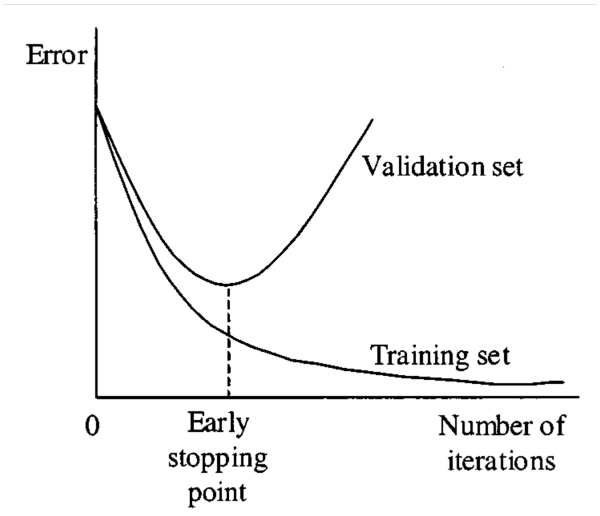

Figure 3.7: Early Stopping Demonstration [7]

4 Generating Data

The Bempp-Cl library was used to generate the data in Python. The general idea was to calculate the far-field scattered pattern from a discretised grid approximating a scattering object. Then the code creates csv files consisting of 400 points of the discretised far-field pattern and the parameters describing the shape (e.g. radii of ellipsoids, lengths of cuboids). Calculating the far-field pattern is an expensive computation and as machine learning (particularly deep learning) requires lots of data to perform well, the generation of data took considerable time. In fact, the first batch of ellipsoids generated took a MacBook Pro 2015 approximately 72 hours to generate 10,000 data points, even considering a mesh size of 0.3 (any mesh size smaller than this would cause the computation to struggle to complete even once). Hence, for the generation of other datasets, access was granted to use the UCL Mathematics DPS machines to generate the data.

Below is the code used to generate the ellipsoid dataset.

Listing 1: Converting Ellipsoid Radii to Voxels

```
import bempp.api
import csv

from scipy.special import sph_harm
import numpy as np
import scipy.io as sio
import io
import random

from scipy.linalg import lstsq
from matplotlib import pyplot as plt

import numpy as np
import matplotlib.cm as cm
from scipy.spatial import Delaunay
from functools import reduce

#defining wavenumber
k=1

@bempp.api.complex_callable
def dirichlet_fun(x, n, domain_index, result):
    result[0] = np.exp(1j * k * x[0])

@bempp.api.complex_callable
def neumann_fun(x, n, domain_index, result):
    result[0] = 1j * k * n[0] * np.exp(1j * k * x[0])
```

```
class DataGenerator:
    def __init__(self,h = 0.3, max_degree =3):
        '''
        h - meshsize
        k - wavenumber
        data - list of all data points
        dirichlet_grid_fun - dirichlet grid function
        neumann_grid_fun - neumann grid function
        '''
        self.h = h

        self.dirichlet_grid_fun = None
        self.neumann_grid_fun = None

        self.data = []
        self.theta = np.linspace(0, 2 * np.pi, 400)

    def generateData(self, n):
        '''
        Generates Data
        '''
        for i in range(n):
            lengths = np.array([random.uniform(2/3, 2) for i
               in range(3)])
            grid = bempp.api.shapes.ellipsoid(r1=lengths[0],
               r2=lengths[1], r3=lengths[2], h = self.h)
            total_field, info, it_count, space = self.
               calculateTotalField(grid)
            db_pattern = self.calculate_db_pattern(space,
               total_field)
            data1 =  db_pattern.tolist() + centres.tolist() +
               lengths.tolist()
            self.data.append(data1)
        return self.data

    def calculateTotalField(self, grid):
        '''Takes a wavenumber k and grid and calculates total
        field.

        Make sure grid is not too fine, otherwise it will take
             too long.

        Returns the total field, info, number of iterations
           and the
        function space (that is calculated from the grid.)
        '''
        space = bempp.api.function_space(grid, "P", 1)
        #print("The space has {0} dofs".format(space.
            global_dof_count))
        identity = bempp.api.operators.boundary.sparse.
            identity(
              space, space, space)
        dlp = bempp.api.operators.boundary.helmholtz.
            double_layer(
              space, space, space, k)
        hyp = bempp.api.operators.boundary.helmholtz.
            hypersingular(
              space, space, space, k)

        #Replacing ntd with 1/ik
        ntd = 1/1j*k

        burton_miller = .5 * identity - dlp - ntd * hyp

        self.dirichlet_grid_fun = bempp.api.GridFunction(space
            , fun=dirichlet_fun)
        self.neumann_grid_fun = bempp.api.GridFunction(space,
            fun=neumann_fun)
        rhs_fun = self.dirichlet_grid_fun - ntd * self.
            neumann_grid_fun

        total_field, info, it_count = bempp.api.linalg.gmres(
              burton_miller, rhs_fun, use_strong_form=True,
              return_iteration_count=True,)
        return total_field, info, it_count, space

    def calculate_db_pattern(self, space, total_field):
        '''
        Takes an argument of Space and Total Field
        Calculates Far-Field Pattern
        Returns the Far-Field Pattern and the respective
        '''
        #bempp.api.global_parameters.assembly.
            potential_operator_assembly_type = 'dense'
        pylab.rcParams['figure.figsize'] = (8.0, 8.0)

        points = np.array([np.cos(self.theta), np.sin(self.
            theta), np.zeros(len(self.theta))])
        dlp_far_field = bempp.api.operators.far_field.
            helmholtz.double_layer(
              space, points, k)
        far_field = dlp_far_field * total_field
        max_incident = np.abs(self.dirichlet_grid_fun.
            coefficients).max()
        radiation_pattern = (np.abs(far_field/max_incident)
            **2).ravel()
        db_pattern = 10 * np.log10(4 * np.pi *
           radiation_pattern)
        return db_pattern

    def writeCSV(self, filepath):
        '''
        Writes the data to a csv with column names for the
           first entry
        '''

        label_dim = 6
        feature_dim = len(self.theta)
        columns = ['feature_'+str(i+1) for i in range(
           feature_dim)] + ['label_'+str(i+1) for i in range(
           label_dim)]

        self.data.insert(0, columns)

        with open(filepath, 'w') as csvFile:
            writer = csv.writer(csvFile)
            writer.writerows(self.data)
        csvFile.close()
        self.data.remove(columns)
        return self.data

for i in range(100):
    dg = DataGenerator(h=0.3)
    dg.generateData(100)
    dg.writeCSV('Ellipsoids/EllipsoidData_%d.csv' % i)
```

To generate datasets for other shapes, the code was easily modified by adjusting the definition of the grid used in the "generateData" function of the "DataGenerator" class. The expensive calculation is in the "calculateTotalField" of the "DataGenerator" class, when the bempp.api.linalg.gmres is called which is an iterative solver similar to the scipy.sparse.linalg.gmres and could possibly reduced by increasing the tolerance of the computation.

The data generated by this code was therefore very clean, uniform and required very little preprocessing due to it not being affected by noise from an inhomogenous medium for example. This compared to most machine learning tasks on real world data is very fortunate. There is obviously very little noise in this data which should also ease the machine learning task as this is far from "real world data". To ensure the data being generated looked sensible, it was important to visualise the data for example in Figure 4.1 - using the Plotly library was especially helpful because of its ability to render interactive 3D plots (much better than Matplotlib).

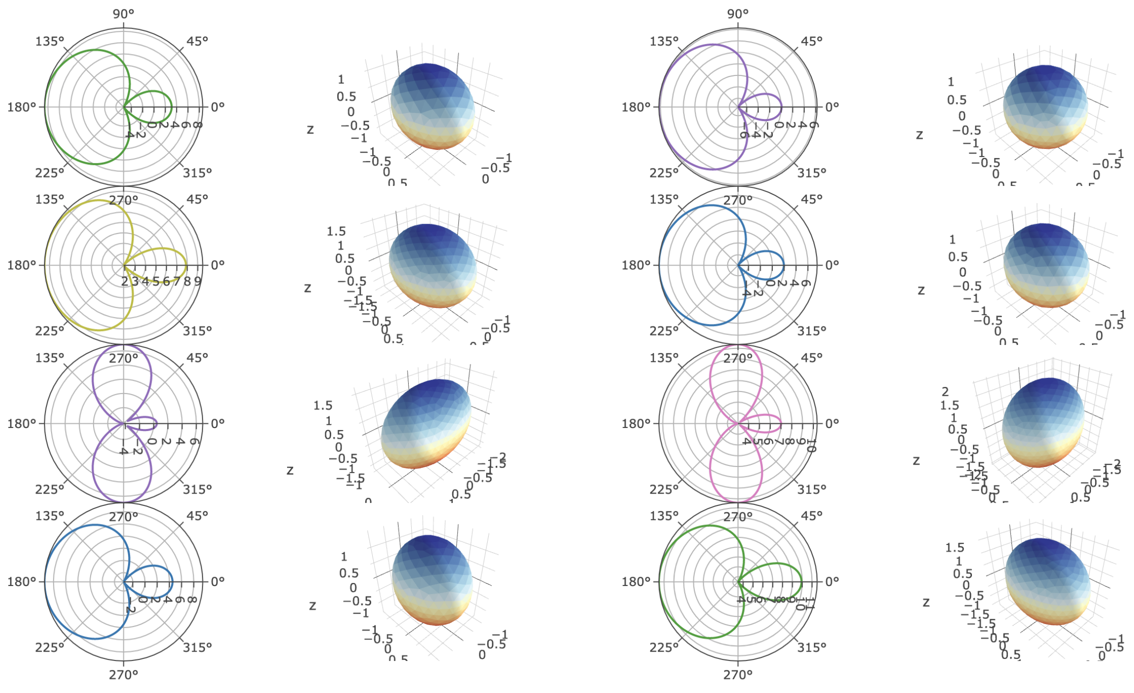
Figure 4.1: Far Field Patterns and there corresponding scattering object

5 Ellipsoid Radii

The first attempt to use Machine Learning was to regress the three radii defined by an ellipsoid from the scattered far-field pattern. The models used were a Neural Network, a Random Forest Classifier and a Support Vector Regression. The data consisted of 400 equally spaced points between 0 and 2π of the far-field pattern of an ellipsoid defined by their 3 radii (the target features) between 1 and 2. 2500 data points were used for training, 2500 for validation and 5000 for a final test set to compare the 3 different models. This is obviously a very large simplification of the task, but was used to gauge how successful ML was going to be.

To optimise training of the SVR and Random Forest, a grid search was used to find optimal hyper-parameters. The hyper-parameters tuned for the Random Forest were:

- N Estimators.
- Max Depth.

and for SVR:

- Regularisation parameter, C.
- Epsilon in the Epsilon SVR parameter.

The Neural Network Architecture was fairly simple. It consisted of:

- a 400 dimension input layer
- a 300 dimension hidden layer
- a 200 dimension hidden layer
- a 100 dimension hidden layer
- a 75 dimension hidden layer
- a 50 dimension hidden layer
- a 25 dimension hidden layer
- a final output layer of dimension 3.

All weights were initialised using a normal distribution and all but the output layer had a Relu activation function. The model was implemented using Keras, and used an Adam optimiser along with the ReduceLROnPlateau method.

The results in Table 5.1 show that ML is capable of this simple approximation of the problem, and even the simpler models performed well. This is also illustrated by Figure 5.1. Evaluation time was included as a performance metric as classical methods are able to achieve a

Model	Training Set MSE	Validation Set MSE	Test Set MSE	Test Set Eval. Time (s)
Random Forest	2.06e-04	1.26e-03	1.46e-03	0.564
SVR	6.16e-05	6.48e-04	7.08e-04	10.2
NN	3.49e-06	5.26e-06	5.52e-06	0.293

Table 5.1: Model Results for Ellipsoid Radii Regression, colour bar indicates log error at each point

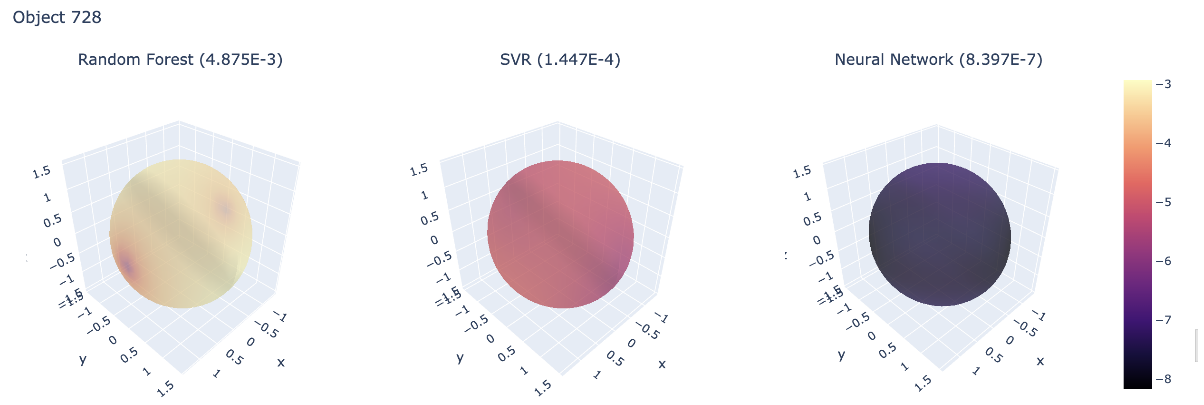

Figure 5.1: An illustration of the log error for each point of the defined ellipsoid

strong accuracy but very slowly, therefore the aspect that machine learning is trying to improve upon is speed. It is therefore evident that the Neural Network's prevalence in terms of accuracy and evaluation time determines that it was the strongest performing model on this task. Additionally, it is evident that although all three models are overfitting slightly, the neural network is overfitting the least.

The Training/Validation Loss Curves 5.2 show that the neural network is slightly overfitting but by referring to 3.7 we can see that this seems similar to an optimum early stopping time.

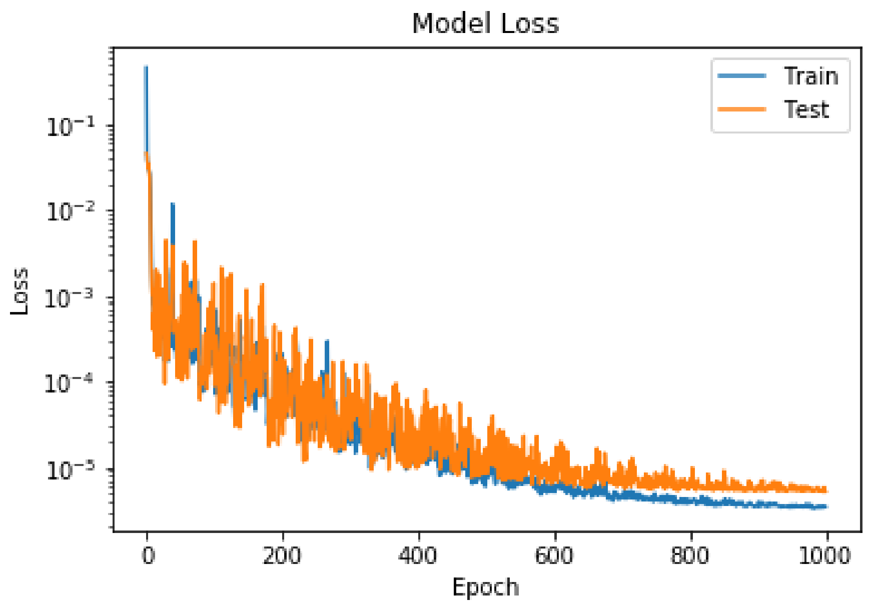

Figure 5.2: Training and Accuracy Loss for Neural Network against number of epochs

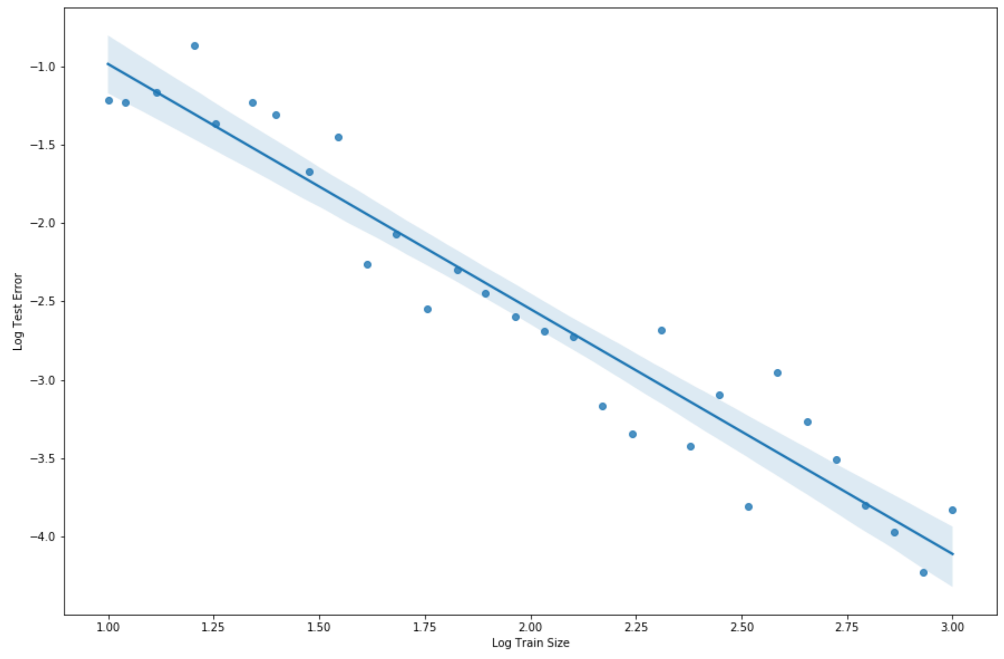

Figure 6.1: Log Training Size vs Log Test Error, with linear interpolation

6 Reducing Dataset Size and Testing Generalisation

6.1 Concerns About Using Too Much Data

6.1.1 Extrapolating Ellipsoid Radii

In the initial attempts a lot of data was used (10,000 datapoints - 2,500 for training, 2,500 for validation and 5,000 for test). There were concerns that the model might be overfitting to the entire dataset. To test this theory, a model with the same architecture was trained for 500 epochs (but also including early stopping based upon validation accuracy) for decreasing training dataset size. Then the test accuracy (MSE) was calculated for each of these models. The results are presented in 6.1.

The results in Figure 6.1 do show that as we decrease dataset size the model performs worse, however, there does not seem to be a catastrophic failure when we remove more and more data. This is supported by a R^2 score of 0.92 for the linear regression fit. Where

$$R^2 = 1 - \frac{\sum_i (y_i - \hat{y}_i)}{\sum_i (y_i - \bar{y})},$$

where y_i is the actual output, $\hat{y}_i$ is the predicted output and $\bar{y}$ is the mean of the actual outputs. The R^2 is a metric to measure the goodness of fit of a linear model with an optimum result of 1.

Although it is not obvious, this is a promising result as the consistency of the decline (shown

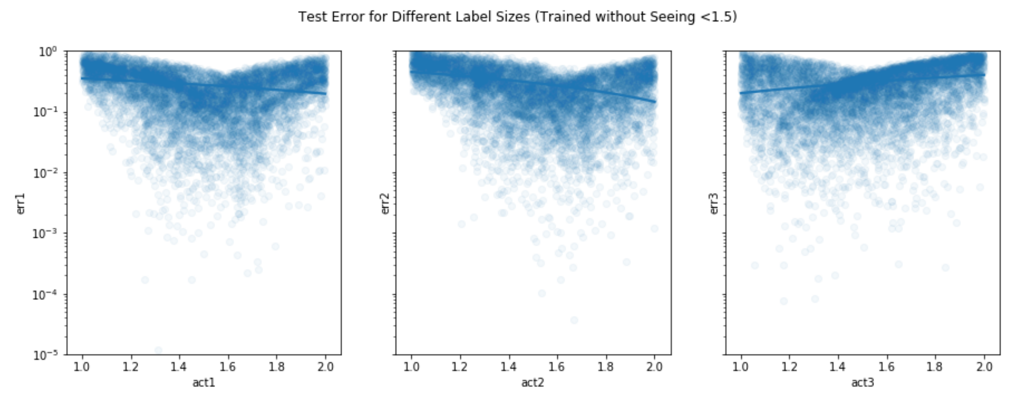

Figure 6.2: The testing error after training a NN on radii less than 1.25 and greater than 1.75

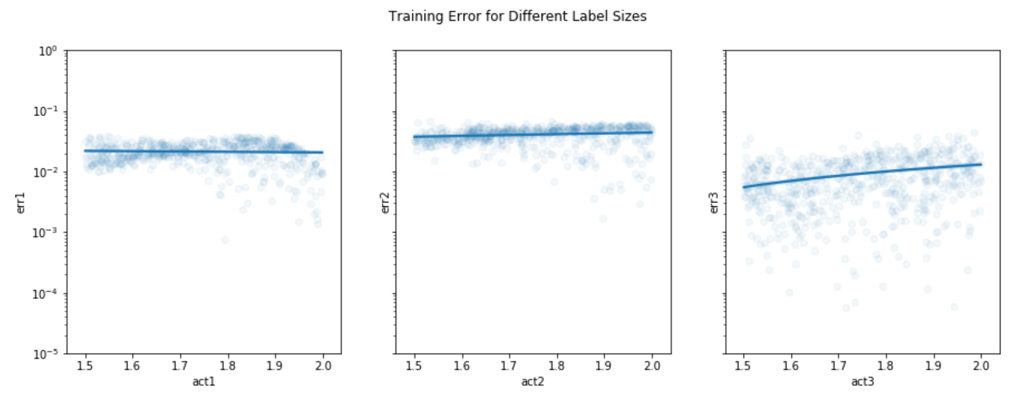

Figure 6.3: The training error after training a NN on radii greater than 1.25 and less than 1.75

by the thin 95% confidence interval in Figure 6.1) is strong. The gradient is approximately -1.6, so it could be said that increasing the dataset size by a factor of 10 would cause a decrease in a factor of approximately 40 for the validation error. This addressed the previous concern of a catastrophic failure for less data. In general, most deep learning tasks mainly struggle due to lacking quality and quantity of data. As the data used here is generated by code, the quality and quantity of it is more than satisfactory and as we use more data for training we predict that the model would perform better.

6.2 Concerns About Not Generalising

It was also important to check how well the model was generalising the function of mapping a scattered pattern to a (simplified) representation of the scattering object. To do this, the model was trained on ellipsoids of radii of certain ranges and then evaluated on a test set of ellipsoids containing radii of length outside this range.

Firstly, to see whether the function could extrapolate it was trained only on ellipsoids where all three radii were greater than 1.75 and less than 1.25. Then it was evaluated on the rest of the dataset. The results show that the model performs poorly on this task, but this is expected as there is no reward for the model to predict outside of the range of 1.25 to 1.75.

The results displayed in Figure 6.2 and 6.3 are scatter plots where the x axis is the length

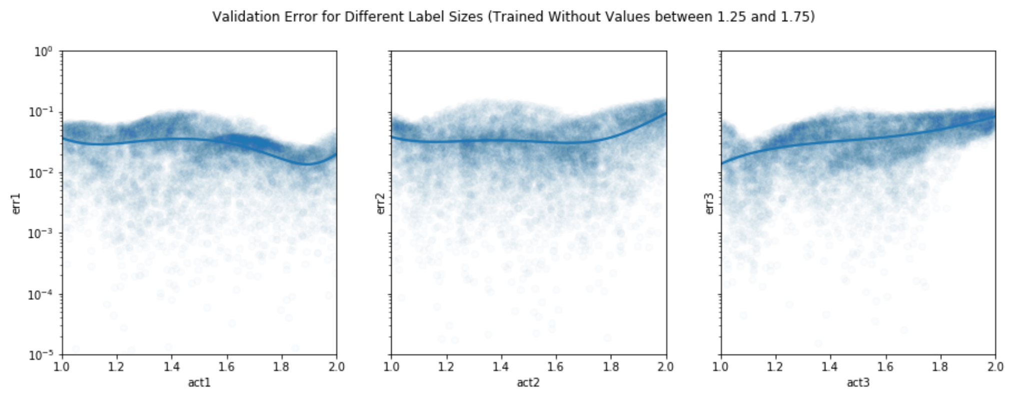

Figure 6.4: The testing error after training a NN on radii less than 1.25 and greater than 1.75

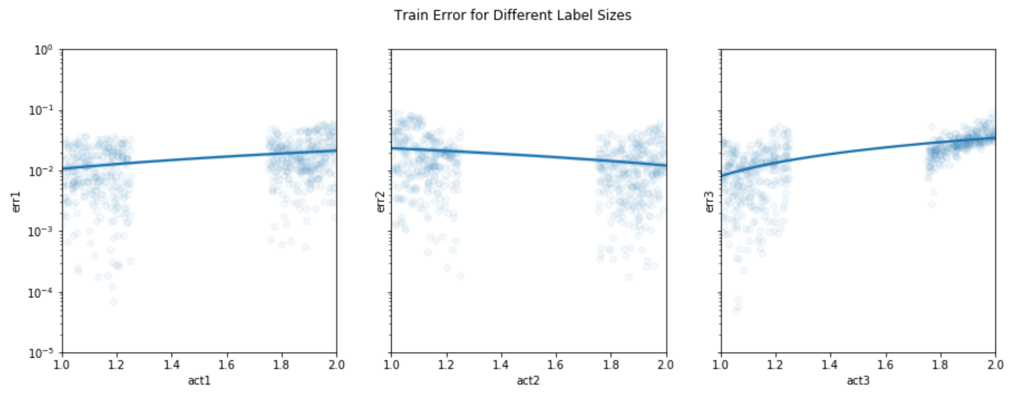

Figure 6.5: The training error after training a NN on radii less than 1.25 and greater than 1.75

radii that the model was trying to predict and the y axis is the error. Figure 6.2 is the testing error and Figure 6.3 is the training error (each column represents one of the three radii). The training error for this task was 0.000818 and the test error was 0.0369. This is a large error but is expected as a model generally won't try and predict a value outside of the range it has seen before because it has never been rewarded for doing so previously. We also tried to see the models ability to interpolate, which was hoped to be more successful.

6.2.1 Interpolation

Similarly, a model with the same architecture was trained only on shapes with all three radii between the range of 1 to 1.25 and 1.75 to 2. The rest of the dataset was used to test the models ability to generalise over the middle range of 1.25 to 1.75. Figure 6.4 and Figure 6.5 are similar graphs to that of Figure 6.2 and Figure 6.3 showing the training and testing results.

We can see that the model performed much better on this interpolation task. The training error was 0.000549 and testing error was 0.000609. This shows the model is slightly overfitting to the values it has seen before but a little bit of overfitting is common in most Machine Learning Models and quite often encouraged as seen in 3.7.

In conclusion, these results give confidence that the model is not overfitting to the data but generalising quite well. It performs worse with less data, but there is no catastrophic blowup or the error for less data which is reassuring. We also tested the ability of the model to interpolate and extrapolate on ranges it was trained on. On the extrapolation task the model performed poorly with severe overfitting, however on the interpolation task the results were far more promising.

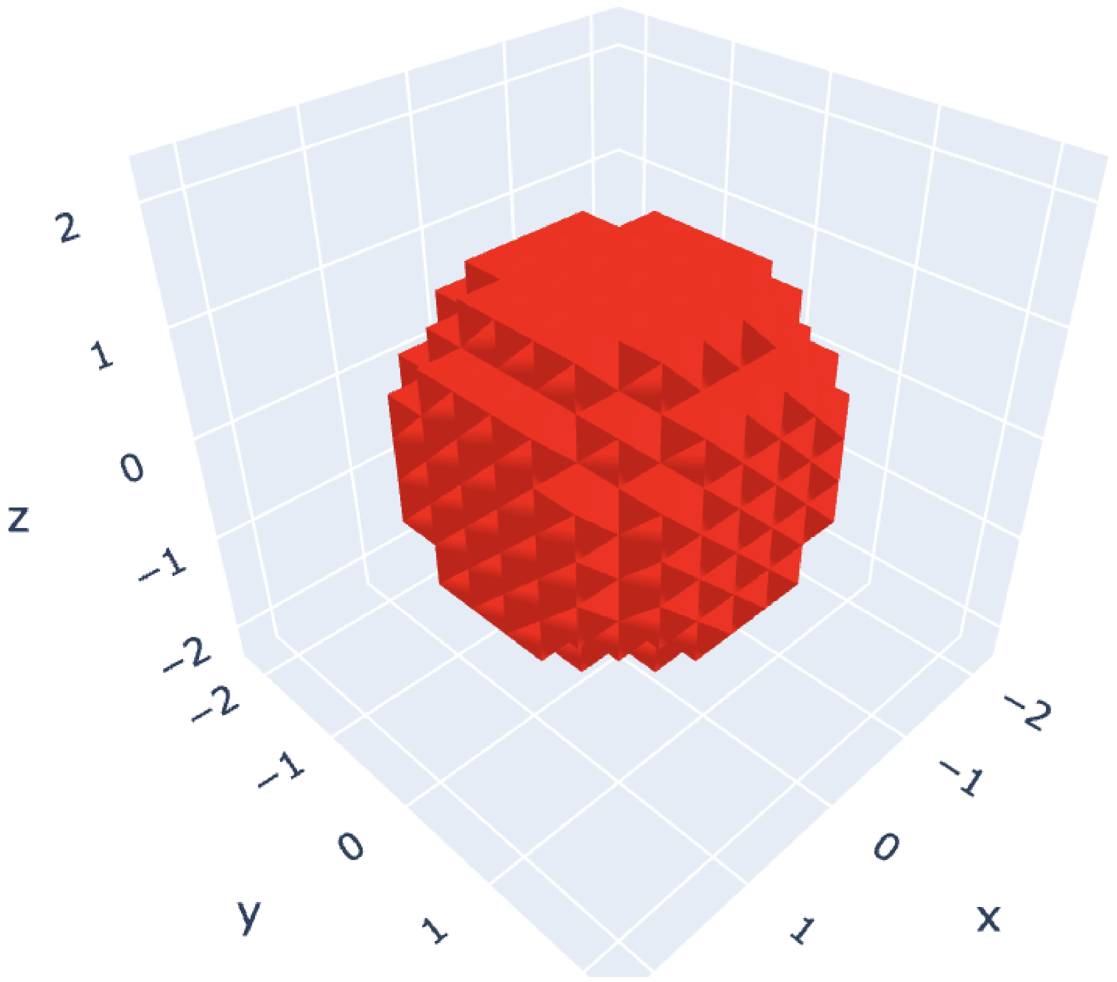

Figure 7.1: A voxel representation of an ellipsoid

7 Ellipsoid Voxels

Voxels represent a value in 3D. It was suggested by Colin Macdonald from The University of British Columbia, that a way of representing the scattering objects would be through voxels. To do this, the space defined by

$$-2 < x < 2,\ -2 < y < 2,\ -2 < z < 2,$$

will be discretised into 0.2 by 0.2 by 0.2 voxels. The model will then try to predict whether the centre of each of these voxels will be inside or outside of the shape (i.e. $11^3 = 1331$ booleans). This makes the predicting challenge obviously more complex and more importantly allows for the model to predict a much wider variety of shapes compared to just ellipsoids. Figure 7.1 is an illustration of a Voxel Representation of an Ellipsoid.

The problem has changed significantly now from a Machine Learning point of view. Instead of being a regression problem, it is now a classification problem; instead of predicting 3 target variables, there are now 1331. In most classification tasks, it is standard to preprocess (normalise) the input variables. I did this using the Standard Scaler which scales data as follows:

$$z = \frac{x-\mu}{\sigma},$$

where x is the original value, μ is the mean of the training samples and σ is the standard deviation of the training samples.

Instead of generating a whole new dataset, the initial dataset of ellipsoids were modified using the following Python Code:

Listing 2: Converting Ellipsoid Radii to Voxels

```python
# Y is a DataFrame of the three radii of ellipsoids.

import numpy as np
import pandas as pd

def insideEllipsoid(radii, point):
  a = radii[0]
  b = radii[1]
  c = radii[2]

  x = point[0]
  y = point[1]
  z = point[2]

  return (x/a)**2 + (y/c)**2 + (z/c)**2 < 1

# Creating The Voxels
gridpoints = []
for z in np.linspace(-2,2,11):
  for y in np.linspace(-2,2,11):
    for x in np.linspace(-2,2,11):
      gridpoints.append((x,y,z))


Y_dots = []

for j in range(10000):
  if j%100==0:
    print(j)
  Y_dots.append([insideEllipsoid(Y.iloc[j], gridpoint) for
     gridpoint in gridpoints])

y_voxels = pd.DataFrame(Y_dots, columns = gridpoints)
```

It can be seen that by modifying the "insideEllipsoid" function, we could similarly find voxel representations of different shapes.

The neural network was of a similar architecture as in the ellipsoid regression task but with the differences being the output layer being of size 1331 (11^3) rather than 3 and the activation at the output layer was a sigmoid function rather than having no activation. This was chosen because the model should predict a probabilistic output between 0 and 1 representing the probability of whether each voxel is inside or outside the scattering object. Additionally, the loss function for the model is binary cross-entropy which is defined as

$$H(p) = -p\log(1-p) - (1-p)\log(p),$$

and was introduced earlier in the machine learning theoretical section.

Again it was important to use some baselines to evaluate the performance of the neural network. First of all a Logistic Regression Model was trained on the same standardized training dataset, which achieved the following:

- Training Accuracy of 70.8%
- Validation Accuracy of 71.6%
- Test Accuracy of 71.1%

These results initially seem quite impressive on the task, however upon further investigation of the precision and recall of the model the failure of the model becomes evident.

7.1 Introducing Precision and Recall

As the task now is a classification task, there are different metrics aside from accuracy that should be considered to assess the model. Take the example of this task, if we try and predict whether a point in our larger grid is inside or outside the shape, if the dataset consisted of small shapes, only a small percentage would be inside the shape and return true. Then, if a model simply predicted that all the voxel points were outside the shape - a nearly 100% accuracy would be achieved yet the model would be obviously failing.

We define when a gridpoint is predicted inside the shape as a positive class and when it is outside as a negative class. For each point the model predicts we define it as true if it was predicted correctly and false if it was predicted incorrectly. Hence, we can define a True Positive (TP) as when the model correctly predicts a point inside the shape and a False Positive (FP) as when a point outside the shape was incorrectly predicted as inside and similarly for True Negatives (TN) and False Negatives (FN).

Precision is defined as $\frac{TP}{TP+FP}$ and could be interpreted as the accuracy of the model on only the points it predicts to be inside the shape. Similarly, recall is defined as $\frac{TP}{TP+FN}$ which could be interpreted as the percentage of points inside the shape which are correctly predicted as inside.

A perfect model would achieve a recall and precision of 1.

Table 7.1: Results of Logistic Regression Model On Ellipsoid Voxels Task

Metric	Train	Validation	Test
Accuracy	0.7076	0.7157	0.7106
Precision	0.1595	0.1559	0.1582
Recall	0.1596	0.1549	0.1581

Table 7.2: Results of Neural Network Model On Ellipsoid Voxels Task

Metric	Train	Validation	Test
Accuracy	0.9992	0.9977	0.9977
Precision	0.9974	0.9926	0.9927
Recall	0.9981	0.9938	0.9939

7.2 Comparing Precision and Recall of Models

Hence it is evident by observing the very poor precision and recall scores displayed in Table 7.1 that the simple Logistic Regression model is not performing well, as expected. Logistic Regression is a linear model and therefore would not be expected to perform well on a non-linear task. However, the neural network performed very well demonstrated by the results in Table 7.2.

The results for the NN are very impressive and it is time to progress on to make the task harder by introducing new shapes. There's a slight overfitting but achieving a near 99.8% accuracy is very promising.

8 Multiple Shape Voxels

8.1 Ellipsoids and Cuboids

The previous strong performance of the voxel task by the machine learning models allows us to move onto a trickier task by predicting voxel representations of different shapes. To do this, a similar dataset to the ellipsoid dataset was created but in this case with cuboids. The cuboids are similarly defined as ellipsoids as they can be characterised by their three lengths. The cuboids were then transformed into their voxel representation using a modified version of the "inside Ellipsoid" function defined earlier on in the paper in Listing 2 to:

Listing 3: Converting Cuboid Length to Voxels

```
def insideCuboid(point, y):

    p0 = y[0]
    p1 = y[1]
    p2 = y[2]

    l0 = y[3]
    l1 = y[4]
    l2 = y[5]

    bool0 = (point[0]>-l0/2) & (point[0]<l0/2)
    bool1 = (point[1]>-l1/2) & (point[1]<l1/2)
    bool2 = (point[2]>-l2/2) & (point[2]<l2/2)

    return bool0 & bool1 & bool2
```

Similarly to the previous task, a regularised logistic regression and a standard feed-forward neural network were used. After the success of the neural network on the previous voxel task, the number of parameters in the model were limited to increase evaluation speed. The structure of the neural network was defined as follows:

- a 400 dimension input layer
- a 100 dimension hidden layer with a ReLu activation
- a final output layer of dimension 1331 with a sigmoid activation

The results of this task were very similar to that of the previous voxel task. The Logistic Regression achieved a higher accuracy, but the poor recall and precision scores still imply a deeper failure of the model. More notably, the neural network maintained impressive results despite limiting the networks architecture and giving a more complicated task. The results for

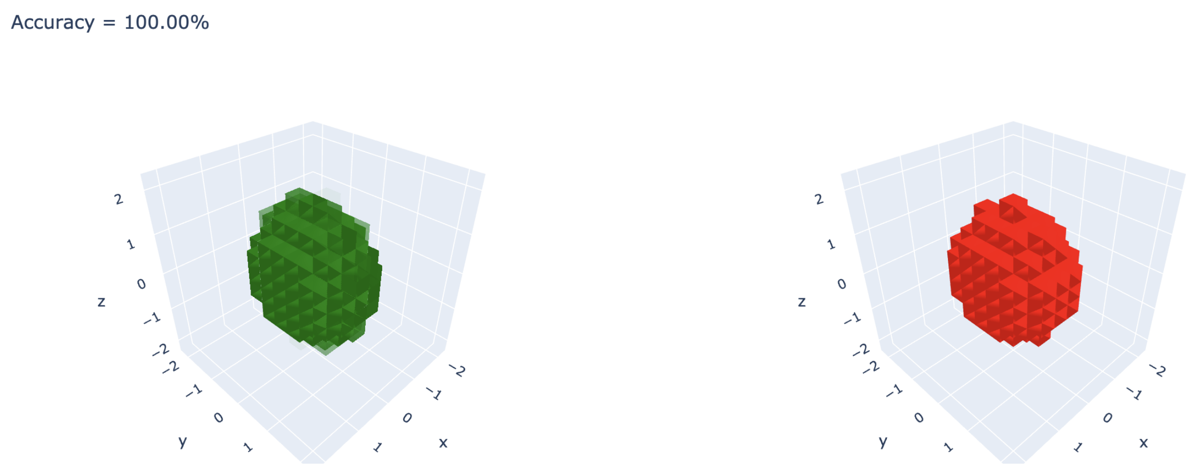

Figure 8.1: Neural Network trained on cuboids and ellipsoids prediction on an ellipsoid (Green represents prediction, Red represents actual)

Metric	Train	Validation	Test
Accuracy	0.829	0.829	0.829
Precision	0.094	0.094	0.094
Recall	0.093	0.095	0.095

Table 8.1: Results of Logistic Regression Model on Cuboids and Ellipsoids

the logistic regression in Table 8.1 and the neural network in Table 8.2 are presented. Additionally, visual representations of the models output are presented in Figure 8.1 and Figure 8.2 again using Plotly, which demonstrates the ability to distinguish between shapes.

8.2 Two separate Shapes

The far-field pattern used as the input to the model is translation independent. Hence, when using the voxel representation of the scattering object, care must be taken to ensure that none of the shapes are translations of another to avoid giving the machine learning an impossible task. When using one shape, this is simple as every shape can be centred around the origin and is what was done when generating the ellipsoid and cuboid datasets. However, when using two shapes it becomes trickier to avoid combinations of shapes which are translations of each

Metric	Train	Validation	Test
Accuracy	0.999	0.998	0.998
Precision	0.996	0.993	0.992
Recall	0.995	0.991	0.990

Table 8.2: Results of Neural Network Model on Cuboids and Ellipsoids

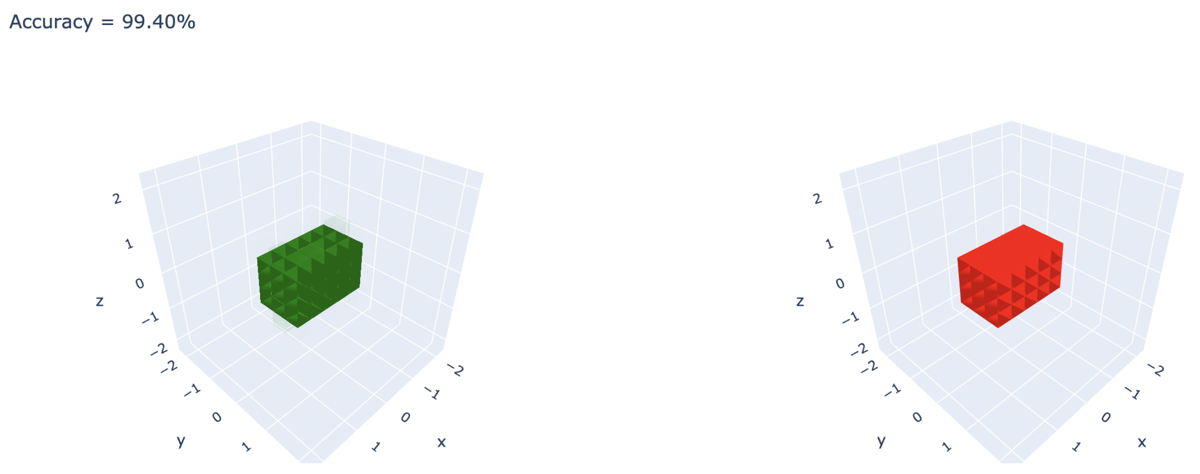

Figure 8.2: Neural Network trained on cuboids and ellipsoids prediction on a cuboid (Green represents prediction, Red represents actual)

other. Additionally, when calculating the scattered pattern for twice the number of shapes at once the calculation takes $2^3 = 8$, times as long. As this came to the tail-end of the project, we were unable to recreate a 10,000 dataset size as we did for the previous ellipsoid and cuboid datasets. The dataset consisted of 5,800 datapoints which were then split for training, validation and testing in a ratio 1:1:2.

To prevent repeats of translation of each other, we created a dataset with two ellipsoids. One was centered at (-1,0,0) and the other varied along the x axis. The code used to describe the grid is presented in Listing 4.

Listing 4: Generating a multiple scattering of two ellipsoids

```
centre1 = np.array([np.random.uniform(2/3,4/3),0,0])
radii1 = np.array([random.uniform(1/3, 2/3), random.
    uniform(1/3, 2), random.uniform(1/3, 2)])

centre2 = np.array([-1,0,0])
radii2 = np.array([random.uniform(1/3, 2/3), random.
    uniform(1/3, 2), random.uniform(1/3, 2)])

grid1 = bempp.api.shapes.ellipsoid(r1=radii1[0], r2=radii1
    [1], r3=radii1[2], origin=centre1, h=self.h)
grid2 = bempp.api.shapes.ellipsoid(r1=radii2[0], r2=radii2
    [1], r3=radii2[2], origin=centre2, h=self.h)

grid = bempp.api.grid.union([grid1, grid2])
```

Table 8.3 presents the results of the Neural Network used in this task of trying to recover the voxels of the two scattering ellipsoids. The results are a little disappointing as the accuracy,

Metric	Train	Validation	Test
Accuracy	0.986	0.980	0.980
Precision	0.890	0.835	0.834
Recall	0.901	0.854	0.852

Table 8.3: Results of Neural Network Model for 2 Scattering Ellipsoids

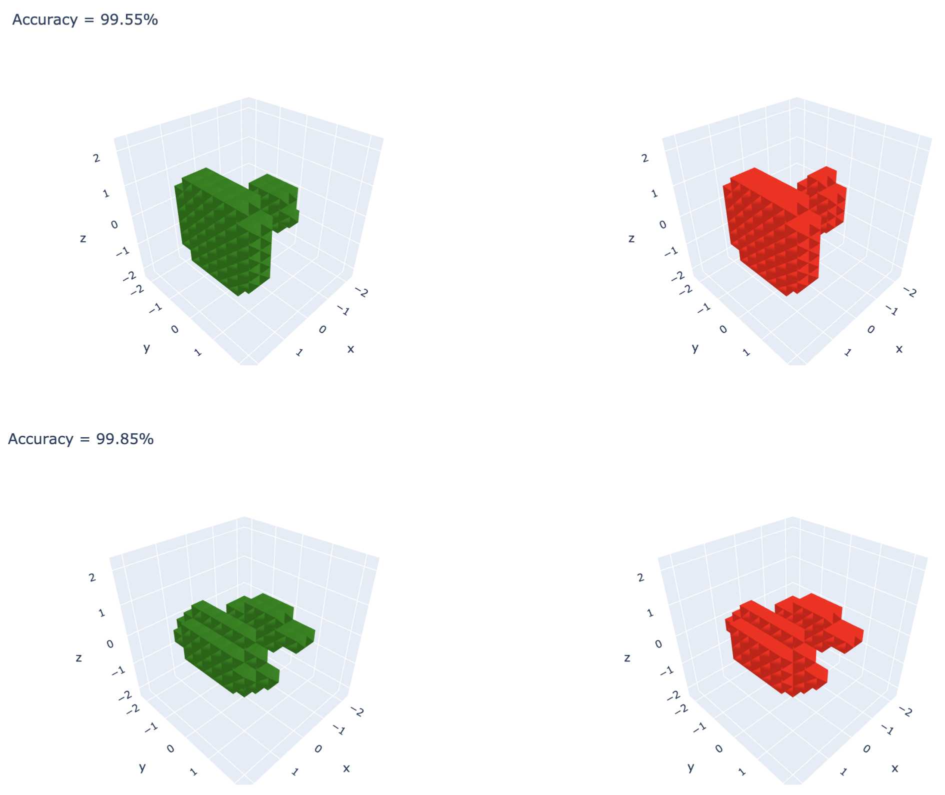

Figure 8.3: Neural Network trained on two scatterers prediction (Green represents prediction, Red represents actual)

precision and recall are not as high as on the previous voxel tasks. However, the results are still high enough that the model distinguishes some key characteristics of multiple scattering objects and performs very well on some examples such as in Figure 8.3. These results are not perfect, however they do suggest that machine learning is capable of performing well on inverse source problems with multiple scattering objects which is a task that current techniques struggle with. We also predict that with more data the model would perform better, although with not as much success as voxels and cuboids as this problem is evidently much harder for the model to generalise over.

9 Discussion and Conclusion

9.1 Summary

In this paper, the inverse source problem was introduced. We used Green's representation formula to derive the boundary integral equations which are used to solve the direct scattering problem. Then the existence of the far-field mapping was shown before developing on to some properties of the far-field mapping. We showed important properties such as the injectivity, the analytical nature and the ill-posedness of the inverse mapping. The ill-posedness was then discussed, particularly on how care must be taken before attempting to find a numerical solution. Decomposition methods are used to overcome this issue of ill-posedness by using a regularised Newton based method that solves the direct scattering problem for each iteration of shape of object. However, this method is very slow so this paper suggests a data-led machine learning approach to approximate the inverse-mapping from a far field pattern to a scattering object in the hope of speeding up the calculation whilst maintaining a strong accuracy of previous classical methods.

Then, some background on machine learning was introduced. First, some applications and general ideas were shown before exploring different models that were implemented in the paper. Then, the logistic regression model was introduced along with decision trees and random forests. From there, we moved onto models in the subset of machine learning called deep learning. Different hyper-parameters were introduced along with some techniques commonly used to achieve better performance.

Subsequently, we went over the code used to generate the data using the Bempp-cl library and talked about how it could be altered for future, more complicated variations of the task. Plotly was also used to visualise the scattering objects and their respective far field patterns to try and gain some insight and intuition into the problem. The first task used to gauge the ability of the machine learning models on was the regression of the three radii defining the shape of a scattering ellipsoid. All the models used in this task performed very well although a shallow neural network was able to achieve the best accuracy and the quickest evaluation time. We also discussed why it was important to use evaluation time as a metric.

Following on from this, we investigated the effect of altering the training set size and values to test the models ability to generalise. Fortunately, the results showed that there was no catastrophic collapse when providing less training data, although the model did perform worse as was expected. The model also showed a strong ability to be able to interpolate over large ranges indicating strong generalisation, although it did struggle when attempting to extrapolate.

The next task was introducing the concept of representing the scattering object using voxels. The benefit of this is it allows the model to predict a wide variety of shapes rather than just ellipsoids. The neural network performed well on this multiple-output classification task achieving around 99.8% test accuracy but also similar scores for recall and precision, particularly compared to the logistic regression benchmark. The model was able to perform well across both ellipsoids and cuboids. Finally, the task for multiple scattering obstacles was also briefly covered. It was found that a neural network could perform well on this task achieving around a 98% test accuracy. Evidently this form of the inverse problem has proved harder for a network to perform on but the results are still good and could be improved by generating a more substantial dataset.

9.2 Discussion of Future Work

This is one of the first documented instances of using machine learning to attempt to solve the inverse source problem. Therefore, there is a lot of room to expand into for future work. Possible extensions include:

- Developing a model on a wider variety of shapes.
- Extensions into electromagnetic scattering.
- Using sound hard objects as scattering objects.
- Experimenting on the effects of the wave number.
- Attempting to find faster/more accurate machine learning models.

The work done could not only allow for faster and more accurate computation on a wider range of scattering objects but could also offer insight into other properties surrounding the inverse scattering problem. This paper demonstrates the capability of machine learning applied to this problem rather than a fully-fledged state of the art model.

References

[1] J. HADAMARD. “Sur les problemes aux derivees partielles et leur signification physique”. In: *Princeton University Bulletin* (1902), pp. 49–52. URL: https://ci.nii.ac.jp/naid/10030001312/en/.

[2] Arnold Sommerfeld. *Partial differential equations in physics / by Arnold Sommerfeld ; translated by Ernst G. Straus.* eng;ger. Sommerfeld, Arnold, 1868-1951. Vorlesungen über theoretische Physik. English v.6. New York: Academic Press, 1949. ISBN: 0126546584.

[3] R. Kress and G. F. Roach. “Transmission problems for the Helmholtz equation”. In: *Journal of Mathematical Physics* 19.6 (1978), pp. 1433–1437. DOI: 10.1063/1.523808. eprint: https://doi.org/10.1063/1.523808. URL: https://doi.org/10.1063/1.523808.

[4] Moshe Leshno et al. “Multilayer feedforward networks with a nonpolynomial activation function can approximate any function”. In: *Neural Networks* 6.6 (1993), pp. 861–867. ISSN: 0893-6080. DOI: https://doi.org/10.1016/S0893-6080(05)80131-5. URL: http://www.sciencedirect.com/science/article/pii/S0893608005801315.

[5] Moshe Leshno et al. “Multilayer feedforward networks with a nonpolynomial activation function can approximate any function.” In: *Neural Networks* 6.6 (1993), pp. 861–867. URL: http://dblp.uni-trier.de/db/journals/nn/nn6.html#LeshnoLPS93.

[6] Leo Breiman. “Random Forests”. In: *Mach. Learn.* 45.1 (Oct. 2001), pp. 5–32. ISSN: 0885-6125. DOI: 10.1023/A:1010933404324. URL: https://doi.org/10.1023/A:1010933404324.

[7] Ramazan Gençay and Min Qi. “Pricing and hedging derivative securities with neural networks: Bayesian regularization, early stopping, and bagging”. In: *Neural Networks, IEEE Transactions on* 12 (Aug. 2001), pp. 726–734. DOI: 10.1109/72.935086.

[8] Yehuda Koren. “The BellKor solution to the Netflix Grand Prize”. In: (Sept. 2009).

[9] Yann LeCun and Corinna Cortes. “MNIST handwritten digit database”. In: (2010). URL: http://yann.lecun.com/exdb/mnist/.

[10] Bengio and Yoshua. *Practical recommendations for gradient-based training of deep architectures*. Sept. 2012. URL: https://arxiv.org/abs/1206.5533.

[11] Diederik Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization.* 2014. URL: http://arxiv.org/abs/1412.6980.

[12] Kaiming He et al. “Deep Residual Learning for Image Recognition”. In: *CoRR* abs/1512.03385 (2015). arXiv: 1512.03385. URL: http://arxiv.org/abs/1512.03385.

[13] Dmilla. *Introduction to Decision Trees (Titanic dataset).* Mar. 2017. URL: https://www.kaggle.com/dmilla/introduction-to-decision-trees-titanic-dataset.

[14] https://github.com/bempp/bempp-cl. 2019.

[15] Will L. Hamilton. *Applied Machine Learning, Lecture 15: Back-propagation.* URL: https://www.cs.mcgill.ca/~wlh/comp551/slides/14-backprop.pdf.