POLITECNICO
MILANO 1863

Numerical Approximation of Inverse Problems for PDEs via Neural Network Augmentation

M.Sc. Thesis by

Dillon Montag
ID: 912846

Supervisor:
Prof. Andrea Manzoni

Program:
Mathematical Engineering

Department of Mathematics
School of Industrial and Information Engineering
Politecnico di Milano
October 2, 2020

Contents

List of Figures

List of Tables

In this thesis, we consider the numerical approximation of inverse problems for linear and nonlinear elliptic PDEs by augmenting them with a neural network to predict unknown or uncertain model coefficients. The neural network acts as a prior for the coefficient and attempts to reconstruct its value from observations of some output quantities of interest related to the PDE solution or from the solution itself. While neural network augmentation for inverse problems has recently been proposed in the literature, we extend the idea to several test cases dealing with diffusion problems and nonlinear elasticity problems. We demonstrate that, under certain conditions, neural networks are highly effective at recognizing many different types of coefficients for these problems. However, we also show that the computational cost of performing these simulations is potentially prohibitive for large-scale applications. To resolve this problem, we attempt to combine neural network augmentation with the reduced basis method with the aim of enhancing computational efficiency. We then discuss the limitations of such an approach and provide some ideas for future applications and further research.

Chapter 1

Introduction

The problem of determining an unknown coefficient from the solution of a partial differential equation (PDE) - known as the inverse problem - has long been both a practically significant and theoretically challenging mathematical problem. From a practical perspective, inverse problems arise whenever one tries to determine the cause of some observable effect. For instance, it is often easy to observe the physical process an object undergoes, such as how much an object bends when a force is applied to it. However, some of the physical properties of the object - like the material it is made from - could be unknown. This is typical in cases that deal with problems like seismic imaging or the identification of material properties of biological tissues. Solving the inverse problem provides information about these unknown properties. From a theoretical perspective, the inverse problem is often challenging to solve. Inverse problems are usually ill-posed, meaning that the solution to them may not be unique or may fluctuate greatly depending on the perturbations of the data. Even more challenging is the attempt to create a comprehensive theory that encompass the various types of inverse problems that exist.

The history of inverse problems starts in the early 1900's with Hermann Weyl, whose theorem - now known as Weyl's law - includes a solution to an inverse problem [20]. Then, in 1929, Viktor Ambartsumian published a seminal paper on finding equations for a given a family of eigenvalues, a paper which became the starting point for the study of a whole family of inverse problems [21]. These problems were then studied in the 1940's by [25] and in the 1960's by [22] and [28] until it became popularized by Tikhonov and Arsenin in the 1970's [40]. Today, there are many published works that provide a comprehensive overview on inverse problems and their challenges (for example, see [4, 30]).

Recently, some authors have combined techniques from mathematics and computer science in an attempt to better understand inverse problems. One such technique, introduced by Berg and Nyström in [9], uses artificial neural networks to guess the solution to the inverse problem. Neural networks are modeled off of neurons in the brain and first began to be studied by McCulloch and Pitts in the early 1940's [29]. Such networks were studied throughout the 1950's by [12], [37], and [38], until Minsky and Papert discovered limitations with the computational machines that processed them in 1969 [31]. Interest in neural networks was rekindled in 1975 when Werbos published his backpropogation algorithm which enabled multi-layer networks to be trained [43]. Since then, neural networks have become an increasingly popular tool in computer science and are behind many current advances in technology, such as self-driving cars [6] and facial recognition [26]. These problems are significantly challenging, and yet neural networks have proven to be an excellent tool to help solve them. Since neural networks have been widely successful at conquering a large variety of complex problems, they could be a promising tool to utilize for solving the inverse problem (for an introduction to artificial neural networks, see [3], [16], and [23]).

In this thesis, we expand on the ideas presented by Berg and Nyström in [9] by using neural networks to recognize the complex functions used to calculate the solutions to different PDEs. We do this by choosing a coefficient within a PDE and representing that coefficient as the output of an artificial neural network. The input to the network is always a vector of vertex points from the mesh that discretizes the domain in which the PDE problem is posed and the output of the network is a vector of points where each point corresponds to the nodal value of the coefficient. This representation means that the coefficient is characterized by the weights of the network. The objective then is for the network to find the correct weights that produce the actual coefficient function when the network is given the vector of vertex points as an input. The advantage of this approach over using a tool like the FEM to estimate the coefficient is many-fold. Firstly, the number of parameters needed by the network to construct the coefficient is much smaller than the number of values required to represent the coefficient at each node of the mesh using finite element functions. As a result, the time it takes to optimize the parameters is faster. Also, the continuity of the approximation produced by the network serves to smooth out any noisy observational data.

The organization of the thesis is as follows: in chapter 2, we introduce the mathematical background needed to understand the inverse problem, the finite element method, and the neural network representation of the coefficient, as well as describe how these concepts have been implemented. In chapter 3, we use neural networks to recognize the (spatially distributed) diffusion coefficient function in the Poisson equation under multiple different scenarios, showing that neural networks are indeed an effective estimator so long as the properties of the network meet certain criteria. In chapter 4, we move to the problem of estimating material properties acting as model inputs in a non-linear elasticity equation - such as Young's modulus for a hyperelastic solid - which allows us to explore a wider variety of increasingly complex scenarios. We show that, once again, neural networks are effective estimators for the coefficient. However, we also show that the computational power needed to run some of the simulations is infeasible for practical applications. Finally, in chapter 5, we attempt to solve this problem by reducing the computational effort needed to solve the inverse problem via the reduced basis method. We show that the reduced basis method can, in fact, be used to reduce the computational complexity of the code. However, we also note some limitations with this approach and provide ideas and insight into future research that could be done to more fully utilize this method.

Chapter 2

Mathematical Background

In this chapter, we introduce the mathematical concepts needed to understand the inverse problem, the finite element method, and neural networks. We begin by introducing the idea of an inverse problem and how to reformulate inverse problems in terms of minimization problems that can be solved using numerical optimization algorithms. We then describe how to approximate the solution to a PDE using the finite element method before moving on to discuss how to represent a PDE coefficient with a neural network. Finally, we describe how these concepts are implemented for the thesis, including the coding framework, packages utilized, and the specifications of the machine running the code.

2.1 The Inverse Problem

Given a function $u(x)$ where $x \in \Omega$ and $\Omega \subset \mathbb{R}^N$, $N \in \{1, 2, 3\}$ is a bounded domain, the inverse problem is to find a $q(x)$ such that the following partial differential equation is satisfied:

$$\begin{aligned} F(u, q) &= 0, && \text{in } \Omega \\ u &= g, && \text{on } \partial\Omega^{\cdot} \end{aligned} \tag{2.1}$$

In equation (2.1), F is a generic differential operator depending on u and q, $\partial\Omega$ is the boundary of Ω, and g is a function which describes the data on the boundary of Ω (for a more rigorous introduction to partial differential equations, see [39]). Almost all inverse problems are ill-posed, meaning that either the solution to the problem is not unique or that a small perturbation in the given data can lead to a large perturbation in the solution (i.e. the problem is not stable). For a more comprehensive overview on inverse and ill-posed problems, see [21].

2.2 The Minimization Problem

The inverse problem can be more effectively formulated in terms of a minimization problem. This is done as follows: pick a $\tilde{q}(x)$ and solve the partial differential equation given by (2.1). This yields a $\tilde{u}(x)$ (which is unique assuming that the PDE problem is well-posed). The difference between $\tilde{u}(x)$ and the actual $u(x)$ can then be defined using the energy functional $J(\tilde{u}) = \frac{1}{2}||u - \tilde{u}||^2$ where $|| \cdot ||$ is a suitable norm (in this thesis, we use the $L^2(\Omega)$ norm). Since $\tilde{u}$ depends on the choice of $\tilde{q}$, this functional can be rewritten as $J(\tilde{u}(\tilde{q})) = \frac{1}{2}||u - \tilde{u}(\tilde{q})||^2$. The minimization problem is to find a q^* that minimizes the functional J subject to the partial differential equation (2.1). In other words, we want to find a q^* such that

$$q^* = \min_{\tilde{q}} J(\tilde{u}(\tilde{q})) \tag{2.2}$$

where $\tilde{u}$ satisfies (2.1).

In practice, the actual value of $u(x)$ is measured using experiments. These measurements are subject to noise, and so the exact value of $u(x)$ is never directly obtained. Instead, measurements always yield a noisy approximation u_{meas} of u. So the functional to minimize is actually

$$J(\tilde{u}(\tilde{q})) = \frac{1}{2}||u_{meas} - \tilde{u}(\tilde{q})||^2 = \frac{1}{2}\int_{\Omega} |u_{meas} - \tilde{u}(\tilde{q})|^2 dx. \tag{2.3}$$

From now on, we will use u instead of $\tilde{u}$ and q instead of $\tilde{q}$ to simplify the notation.

2.3 The Finite Element Method

In general, the PDE (2.1) has to be solved numerically. In this thesis, this will be done through the finite element method (FEM). The general idea behind the FEM is to discretize the domain using a computational mesh and then to approximate the solution u to the PDE on each element of the mesh, typically with some sort of polynomial (e.g. a linear polynomial). The general steps behind the FEM are as follows:

1. Select a space of test functions, usually denoted V, to whom the solution u of the PDE belongs. This is often a space of piecewise polynomials that satisfy the boundary conditions of the original PDE problem.

2. Multiply the differential equation by a test function $v \in V$ and integrate over the domain Ω. Integrate any second order derivatives by parts. This improves the regularity of the PDE by removing one order of the derivative from the overall equation. This new equation is known as the *weak formulation* of the problem.

3. Interpret the weak formulation as an abstract variational problem by defining the appropriate bilinear forms and functional spaces.

4. Discretize the abstract variational problem using the Galerkin method by defining a finite-dimensional subspace V_h of V.

5. Using a basis of V_h, transform the Galerkin problem into an algebraic system that can be solved efficiently using one of many numerical methods.

This process will be applied to the problems in the following chapters. For a more comprehensive introduction to the FEM, see [36].

2.4 Neural Network Representation of q

In equation (2.1), we can take the function q and represent it as the output of a neural network (this is possible since a neural network can approximate any continuous function within an arbitrary accuracy bound [18]). More specifically, let q be parameterized by W and b where W is a collection of $\mathbb{R}^{M\times N}$, $M, N \in \mathbb{N}$ matrices of weights and b is a collection of bias vectors in $\mathbb{R}^N$. The coefficient $q(x; W, b)$ is then computed by feeding the domain forward through the neural network. In other words,

$$\begin{aligned} q &= \sigma_L(z^L) \\ z^L &= W^L \sigma_{L-1}(z^{L-1}) + b^L \\ &\vdots \\ z^2 &= W^2 \sigma_1(z^1) + b^2 \\ z^1 &= W^1 x + b^1 \end{aligned} \tag{2.4}$$

where $L \in \mathbb{N}$ is the number of layers in the neural network, W^l, $l \in \{1, 2, ..., L\}$ is the weight matrix that connects layers l and $l-1$, b^l is the vector of biases for layer l, and σ_l is the activation function for layer l.

With this new representation, the minimization problem (2.2) is to find a collection of weight matrices W^* and a collection of biases b^* such that

$$W^*, b^* = \min_{W,b} J(u(q)) \tag{2.5}$$

where u satisfies equation (2.1).

Note that an iterative method (like the BFGS algorithm) is used to solve equation (2.5). This iterative algorithm starts with an initial guess for W and b, and then at each iteration it uses the gradient of (2.3) with respect to the parameters to derive a new, better guess for W and b. This process is equivalent to training the neural network where the loss function is given by (2.3). Note that algorithms like BFGS can be used in these scenarios because the number of parameters of the network (i.e. the number of weights and biases) is relatively small when compared to the dimension of a finite element space. This is one of the main reasons why we represent q with a neural network instead of searching for q in some finite element space.

2.5 Implementation of the Minimization Problem

We solve the minimization problem (2.5) using Python3 (and PyCharm as the IDE) [41] with the FEniCS package [2] to solve the forward PDE problem via the FEM and the dolfin-adjoint package [32] to calculate the gradient of the functional J. The BFGS optimizer from the SciPy package [42] is used to solve the optimization problem (2.5) and the norm of the gradient is always used as the convergence criterion. The NumPy [34] package is used to generate any random values and perform the various mathematical calculations needed in the code. Finally, the PyTorch package [35] is used to construct any neural networks needed and to compute the gradients of the networks. In this project, the sigmoid activation function is used for every layer in all the neural networks except the output layer where a linear activation function is used. All visualizations shown are made either with FEniCS's plotting engine or with ParaView [5]. The code is run on a 4 core machine with a 6MB cache and 8GB of RAM. The operating system used is the 64-bit version of Windows 10.

Combining all of the concepts mentioned in this chapter, we now provide a general overview of how these tools work together in order to recognize the desired coefficient. The workflow for any problem follows the same overarching pattern:

1. Use PyTorch to create a neural network with the desired number of layers and nodes. Randomly initialize the parameters of the network.
2. Create a mesh and define the abstract variational problem and appropriate functional spaces using FEniCS.
3. Decide on the value of the coefficient to be predicted by the neural network and perform a forward solve of the PDE system (2.1) with FEniCS using this coefficient. Store the resulting u value.
4. If desired, add noise to u. After this step, u is now the u_{meas} mentioned in equation (2.3).
5. Define a function that takes the parameters of the neural network as an input and outputs the value of the energy functional (2.3). This function should update the parameters of the neural network using PyTorch, feed the vector of vertices forward through the updated network to obtain q, and then perform a forward solve of the PDE system (2.1) using this new q to obtain a new value of u. The value of J is then calculated from this new value of u. This can be done in a few lines of code using FEniCS.
6. Define a function that takes the parameters of the neural network as an input and outputs the Jacobian of the energy functional (2.3) with respect to the network parameters. This is the most involved step because it requires the computation of

$$\frac{dJ}{dp} = \frac{\partial J}{\partial u}\frac{\partial u}{\partial q}\frac{\partial q}{\partial p} + \frac{\partial J}{\partial q}\frac{\partial q}{\partial p} = \left(\frac{\partial J}{\partial u}\frac{\partial u}{\partial q} + \frac{\partial J}{\partial q}\right)\frac{\partial q}{\partial p}$$

   where p is any of the weight or bias parameters of the network. The factor $\frac{\partial J}{\partial u}\frac{\partial u}{\partial q} + \frac{\partial J}{\partial q}$ is computed by using FEniCS together with dolfin-adjoint, while the factor $\frac{\partial q}{\partial p}$ is computed using PyTorch.
7. Pass the 2 functions defined in the steps above, along with the initial parameters of the network, to an iterative numerical solver, such as the BFGS algorithm. The algorithm uses these two user-defined functions to solve the minimization problem (2.5).

Chapter 3

Numerical Approximation of the Inverse Problem for the Poisson Equation

In this chapter, we consider the numerical approximation of the inverse problem that involves identifying the diffusion coefficient of the Poisson equation from partial measurements of its solution. The Poisson equation is widely used in physics to describe phenomena such as the potential field given by an electrical charge or the heat distribution over a surface. Not only is the equation widely applicable, but it is also relatively simple with respect to other partial differential equations, making it a logical problem from which to begin the explorations of the effects of neural network augmentation. Berg and Nyström have explored this case in [9] when q is relatively simplistic, the results of which we successfully reproduce. We then extend their ideas to new cases and draw some new conclusions. We show below that neural networks are good predictors of the coefficient in both the 1D and 2D cases, provided that the network has a sufficient number of parameters to correctly capture the complexity of the coefficient function. We also demonstrate that these results can hold when the observations of the solution of the PDE are noisy or limited. Finally, we reveal some patterns that connect the neural network architecture to the efficacy of the network prediction.

3.1 Weak and Algebraic Formulations of the Poisson Problem

Given a domain $\Omega \subset \mathbb{R}^N$ where $N \in \{1, 2, 3\}$, the Poisson problem is to find a $u = u(x)$ such that

$$\begin{aligned} -\nabla \cdot (q(x)\nabla u(x)) &= f(x), \qquad x \in \Omega \\ u(x) &= g(x), \qquad x \in \partial\Omega \end{aligned} \tag{3.1}$$

where $\partial\Omega$ is the boundary of Ω and q, f, and g are known functions. The inverse problem is to find a $q(x)$ that satisfies (3.1) given (partial) measurements of u where both f and g are known.

To derive the weak formulation of the problem, we first decide to look for the solution u to (3.1) in the functional space $V = H_0^1(\Omega)$. We then multiply (3.1) by a test function $v \in V$ and integrate over the domain Ω. This yields

$$-\int_\Omega [\nabla \cdot (q\nabla u)]v \; dx = \int_\Omega fv \; dx. \tag{3.2}$$

Integrating the first term by parts, we obtain

$$\int_\Omega q \cdot \nabla u \cdot \nabla v \; dx = \int_\Omega fv \; dx \tag{3.3}$$

(note that none of the scenarios in this chapter use Neumann boundary conditions). Since we want to search for u in the space $H_0^1(\Omega)$, we want $u = 0$ on the Dirichlet boundary. However, since $u = g$ on the Dirichlet boundary, we introduce the transformation $u = u_0 + R_g$ where

$$\begin{aligned} u_0 &= u \qquad \text{in } \Omega \\ u_0 &= 0 \qquad \text{on } \partial\Omega \end{aligned}$$

and $R_g \in H^1(\Omega)$ is a suitable continuous function such that $R_g|_{\Gamma_D} = g$ (R_g is called a *lifting function*). By substituting this u into (3.3), we obtain

$$\int_\Omega q \cdot \nabla u_0 \cdot \nabla v \; dx + \int_\Omega q \cdot \nabla R_g \cdot \nabla v \; dx = \int_\Omega fv \; dx. \tag{3.4}$$

The weak formulation of the problem is then: Find a $u_0 \in V$ such that

$$\int_\Omega q \cdot \nabla u_0 \cdot \nabla v \; dx = \int_\Omega fv \; dx - \int_\Omega q \cdot \nabla R_g \cdot \nabla v \; dx \qquad \forall\, v \in V. \tag{3.5}$$

We now introduce the bilinear form

$$a(u, v) = \int_\Omega q \cdot \nabla u \cdot \nabla v \; dx$$

and the linear functional

$$F(v) = \int_\Omega f v \; dx - a(R_g, v).$$

The abstract variational formulation of the problem is then: Find a $u_0 \in V$ such that

$$a(u_0, v) = F(v) \qquad \forall \; v \in V. \tag{3.6}$$

We now want to discretize (3.6) in order to approximate u_0 using the Galkerin-Finite Element method. To do this, we select a subspace $V_h \subset V$ where $\dim(V_h) = N_h < \infty$. The Galerkin problem then reads: Find a $u_h \in V_h$ such that

$$a(u_h, v_h) = F(v_h) \qquad \forall \; v_h \in V_h. \tag{3.7}$$

Since $N_h < \infty$, we can find a basis for the space V_h. Let $\{\phi_i\}_{i=1,\dots,N_h}$ denote such a basis. Since all functions in V_h are linear combinations of the basis functions, it is sufficient to show that

$$a(u_h, \phi_i) = F(\phi_i), \qquad i = 1, 2, ..., N_h. \tag{3.8}$$

Also, since $u_h \in V_h$, u_h can be written as a linear combination of the basis functions. More specifically,

$$u_h(x) = \sum_{j=1}^{N_h} u_j \phi_j(x)$$

where the coefficients $u_j \in \mathbb{R}$ are unknown. Substituting this expansion of u_h into (3.8) yields

$$a\Big(\sum_{j=1}^{N_h} u_j \phi_j(x), \phi_i\Big) = F(\phi_i), \qquad i = 1, 2, ..., N_h \tag{3.9}$$

which is equivalent to

$$\sum_{j=1}^{N_h} u_j a(\phi_j, \phi_i) = F(\phi_i), \qquad i = 1, 2, ..., N_h. \tag{3.10}$$

Let A be the matrix with the elements $A_{ij} = a(\phi_j, \phi_i)$, $\boldsymbol{f}$ be the vector with components $f_i = F(\phi_i)$, and $\boldsymbol{u}$ be the vector with components u_j. Then equation (3.10) can be written as the algebraic system

$$A\boldsymbol{u} = \boldsymbol{f}. \tag{3.11}$$

This is the system that will be solved in order to obtain the approximation of u_0. Once u_0 is obtained, the original u can be reconstructed using the fact that $u = u_0 + R_g$.

Let $\mathcal{T}_h$ denote the set of elements in the mesh. For all the problems we consider in this chapter, we will let $V_h = \{v_h \in C^0(\bar{\Omega}) \mid v_h|_k \in \mathbb{P}_1 \; \forall \; k \in \mathcal{T}_h \text{ and } v_h = 0 \text{ on } d\Omega\}$ (i.e. V_h is the set of linear continuous piecewise polynomials on each element of the mesh that vanish on the boundary).

3.2 Estimating the Diffusion Coefficient of the Poisson Equation for the Dimension $N = 1$

We first consider the 1D Poisson equation given by

$$\begin{aligned} -(qu_x)_x &= f(x), \qquad & x \in (0,1) \\ u(0) &= g_0, \qquad & u(1) = g_1 \end{aligned} \tag{3.12}$$

where u, f, g_0, and g_1 are known and $q(x) : \mathbb{R} \to (0, \infty)$ is the coefficient to be estimated. To be able to reproduce the results shown in [9], we consider the same discretization of the domain $(0,1)$ into 101 uniform elements and the resulting 100 vertices.

We solve the forward problem (3.12) by solving the algebraic system (3.11). When solving the minimization problem, the BFGS algorithm stops iterating when the gradient of the error functional is less than 10^{-6}. For some tests, a noise value δr is added to each of the interior data points where $\delta \in [0,1]$ and $r \in \mathcal{N}(0,1)$ (the data on the boundary is always exact). The noise is generated using numpy's random generator which is seeded with the value of 2 to ensure the results are reproducible. The neural network used has 1 input layer, 1 hidden layer with 3 nodes, and 1 output layer. With this network structure, 10 parameters characterize $q(x)$ and need to be optimized. Unless otherwise specified, all measurements of u in this section are noise-free and taken on the entire domain.

3.2.1 Predicting Continuous $q(x)$ Functions

To test the validity of our implementation, we first examine the simple case from [9] where $u(x) = \sin^2(2\pi x)$, $f(x) = -8\pi^2 \cos(4\pi x)$, and $g_0 = g_1 = 0$. In this scenario, the solution to the inverse problem is $q(x) = 1$. This is the value the neural network will attempt to reconstruct.

As shown by figures 3.1 and 3.2, the network is highly accurate in its prediction of q.

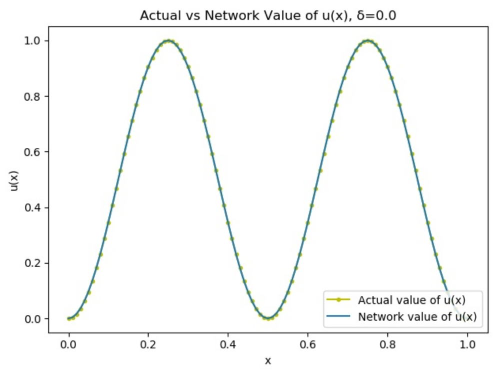

Figure 3.1: Neural network value of $u(x)$ when $q(x) = 1$.

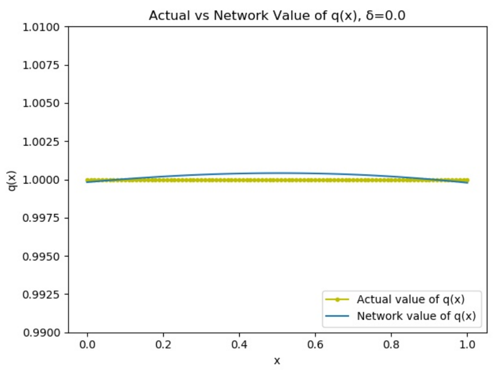

Figure 3.2: Neural network reconstruction of $q(x)$.

To more rigorously quantify the difference between the two lines in the graphs, we compute the errors of u and q in the $L^2(\Omega)$ norm. The low error values in table 3.1 confirm that the prediction of q is indeed accurate.

We now reproduce the results from [9] for the most complicated continuous coefficient to further confirm the accuracy of our implementation. For this scenario, the value of $u(x)$ is the same, but now $f(x) = -2\pi^2[2(\sin(2\pi x) + 2)\cos(4\pi x) + \sin(4\pi x)\cos(2\pi x)]$. In this case, the solution to the inverse problem is $q(x) = 1 + \frac{1}{2}\sin(2\pi x)$.

Figures 3.3 and 3.4 once again show that the neural network is able to predict q with a high amount of accuracy. This is also confirmed by table 3.1 which shows low error values for both u and q. These are the same results reported in [9].

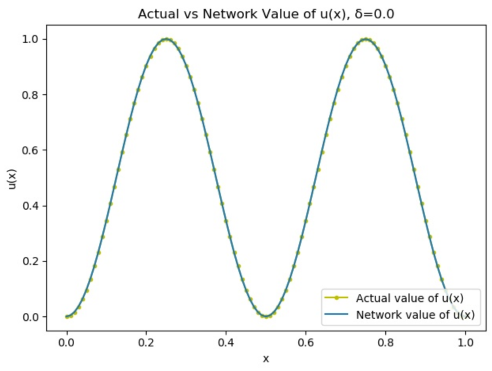

Figure 3.3: Neural network value of $u(x)$ when $q(x) = 1 + \frac{1}{2}\sin(2\pi x)$.

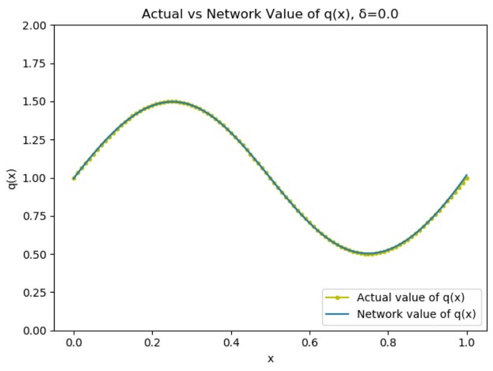

Figure 3.4: Neural network reconstruction of $q(x) = 1 + \frac{1}{2}\sin(2\pi x)$.

$q(x) = 1$					$q(x) = 1 + \frac{1}{2}\sin(2\pi x)$			
#it	Time	$\|\|u - \hat{u}\|\|$	$\|\|q - \hat{q}\|\|$		#it	Time	$\|\|u - \hat{u}\|\|$	$\|\|q - \hat{q}\|\|$
17	1sec	1.25e-04	2.77e-04		379	21sec	3.10e-04	2.89e-03

Table 3.1: Performance of the neural network for the 1D Poisson equation with continuous coefficients.

Note that in table 3.1 above, "#it" is the number of iterations it takes for the BFGS algorithm to converge and "Time" is the time it takes for the BFGS algorithm to converge on a PC with the specifications given in section 2.5.

We now want to confirm the results obtained by our code match those in [9] when looking at noisy observations of u. To perform this test, we take the most complicated continuous coefficient $q(x) = 1 + \frac{1}{2}\sin(2\pi x)$ and try to estimate it when $\delta = 0.05$. Figures 3.5-3.6 below demonstrate the ability of the neural network to recognize q in this case. Table 3.2 further confirms the ability of the network to estimate q when given noisy observations of u. All of these results verify those found in [9], which also indicates that our code is performing as expected.

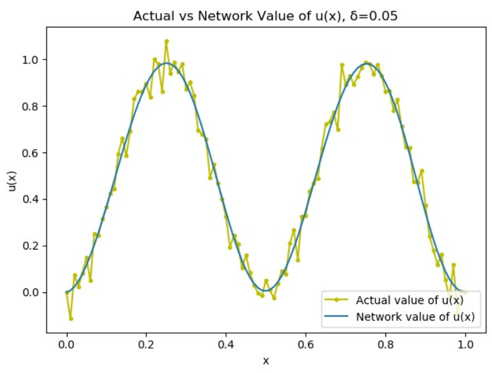

Figure 3.5: Neural network value of $u(x)$ when $q(x) = 1 + \frac{1}{2}\sin(2\pi x)$ and $\delta = 0.05$.

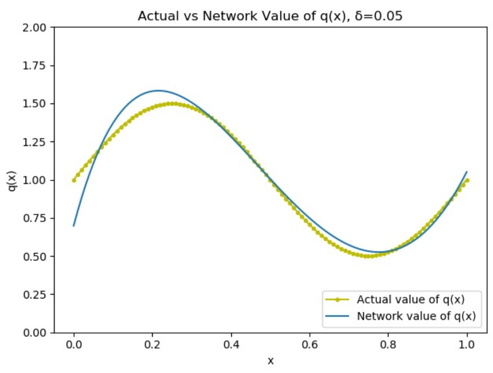

Figure 3.6: Neural network reconstruction of $q(x) = 1 + \frac{1}{2}\sin(2\pi x)$ when $\delta = 0.05$.

$q(x) = 1 + \frac{1}{2}\sin(2\pi x)$, $\delta = 0.05$			
#it	Time	$\|\|u - \hat{u}\|\|$	$\|\|q - \hat{q}\|\|$
814	52sec	1.08e-02	6.26e-02

Table 3.2: Performance of the neural network for the 1D Poisson equation with a continuous coefficient and a noise value of $\delta = 0.05$.

3.2.2 Predicting Discontinuous $q(x)$ Functions

Berg and Nyström have demonstrated the ability of a neural network to successfully predict $q(x)$ when $f = 10$, $g_1 = g_2 = 0$, and

$$q = \begin{cases} 0.5, & 0 \leq x < 0.5 \\ 1.5, & 0.5 \leq x \leq 1 \end{cases} \tag{3.13}$$

[9] (note that, in this case, u is calculated *a priori* via the FEM using this q. The value of q is then "forgotten" and the neural network attempts to reconstruct it). When exploring more complicated discontinuities, we found that the network used by Berg and Nyström is not sufficient to capture the complexity of the $q(x)$ function. For example, the case when $f = 10, g_1 = g_2 = 0$, and

$$q = \begin{cases} x + 1, & 0 \leq x < 0.5 \\ -x + 1, & 0.5 \leq x \leq 1 \end{cases} \tag{3.14}$$

is shown below. Figure 3.7 shows the inability of the network to accurately capture $q(x)$ with only 3 nodes in the hidden layer. However, when the number of nodes in the hidden layer is increased to 9, the network is able to accurately capture the behavior of $q(x)$, as shown in figure 3.8. This is confirmed by table 3.3 below, which shows that errors are smaller when using 9 nodes in the network as opposed to 3 nodes.

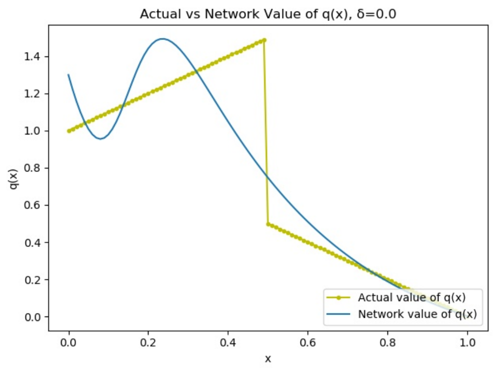

Figure 3.7: Neural network value of $q(x)$ with 3 hidden layer nodes.

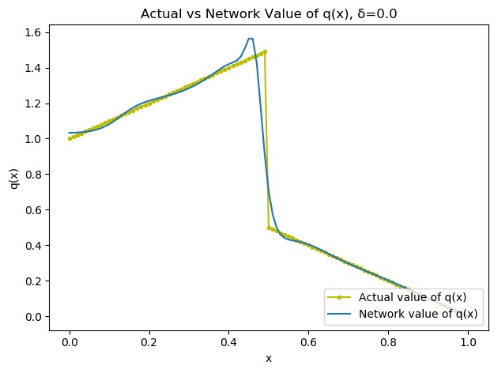

Figure 3.8: Neural network value of $q(x)$ with 9 hidden layer nodes.

$q = \begin{cases} x+1, & 0 \le x < 0.5 \\ -x+1, & 0.5 \le x \le 1 \end{cases}$								
3 hidden layer nodes					9 hidden layer nodes			
#it	Time	$\|\|u-\hat{u}\|\|$	$\|\|q-\hat{q}\|\|$		#it	Time	$\|\|u-\hat{u}\|\|$	$\|\|q-\hat{q}\|\|$
1531	142sec	5.89e-02	2.09e-01		1505	130sec	2.85e-03	6.20e-02

Table 3.3: Performance comparison between a 3 and 9 hidden layer node network for the 1D Poisson equation with a discontinuous coefficient - example 1.

Note that, at each iteration of the numerical optimization procedure, a finite element problem of size N_h needs to be solved where N_h is the dimension of the finite element space, and so the time taken to run the algorithm is directly related to the size of N_h. This example provides, as far as we are aware, the first numerical evidence to support the claim in [9] that the structure of the neural network impacts the performance of the network when solving the inverse problem.

The same situation arises for the case when $f = 10$, $g_1 = g_2 = 0$, and

$$q = \begin{cases} \sin(4\pi x) + 1, & 0 \le x < 0.5 \\ -x + 1, & 0.5 \le x \le 1 \end{cases}. \tag{3.15}$$

For this case, the oscillating nature of $q(x)$, the discontinuity, and the change in the slope of the function at the discontinuity point all make the coefficient highly irregular and so difficult for a neural network to predict. While a network with 3 nodes in the hidden layer traces the general shape of the coefficient, a network with 9 nodes in the hidden layer drops the error by an entire order of magnitude as shown in figure 3.10 below. This is confirmed by the error chart in table 3.4, which shows that the errors for u and q are indeed lower for a 9 hidden layer node network.

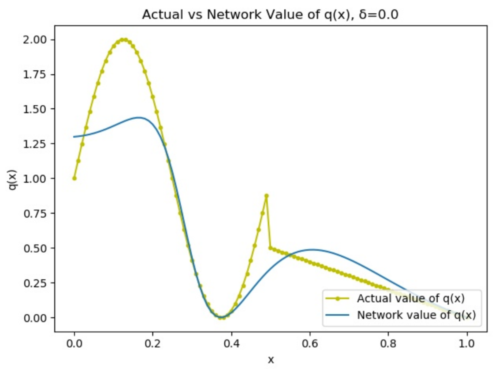

Figure 3.9: Neural network value of $q(x)$ with 3 hidden layer nodes.

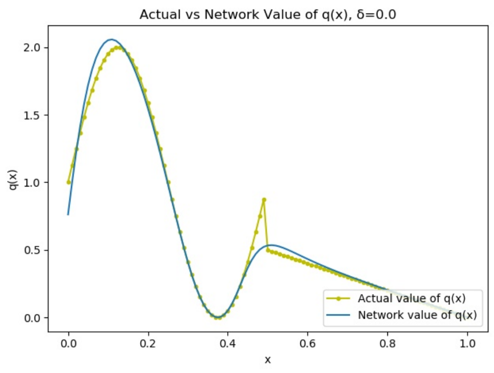

Figure 3.10: Neural network value of $q(x)$ with 9 hidden layer nodes.

$q = \begin{cases} \sin(4\pi x) + 1, & 0 \le x < 0.5 \\ -x + 1, & 0.5 \le x \le 1 \end{cases}$								
3 hidden layer nodes					9 hidden layer nodes			
#it	Time	$\|\|u-\hat{u}\|\|$	$\|\|q-\hat{q}\|\|$		#it	Time	$\|\|u-\hat{u}\|\|$	$\|\|q-\hat{q}\|\|$
1970	206sec	5.44e-02	2.18e-01		1215	106sec	5.53e-03	6.31e-02

Table 3.4: Performance comparison between a 3 and 9 hidden layer node network for the 1D Poisson equation with discontinuous coefficient - example 2.

Once again, this example provides numerical evidence to show that the structure of the neural network can have a significant impact on the network's predictive ability. However, as the number of parameters in the network increases, it becomes possible that the network begins to over-fit the data. More research needs to be done to determine the ideal structure of a network for a given problem *a priori*.

We now take the coefficient $q = \begin{cases} \sin(4\pi x) + 1, & 0 \leq x < 0.5 \\ -x + 1, & 0.5 \leq x \leq 1 \end{cases}$ from (3.15) and examine what happens when noise is added to the original measurements of u (we use the 9 hidden layer node network since this network gives the best prediction of q). We let δ increase in increments of .10 until $\delta = 0.50$. The results are summarized in table 3.5 below. As shown in figures 3.11-3.12, the neural network approximates noisy functions with continuous ones, which serves to smooth out the noise. This is a desirable result - it is likely that any measurements of u will be noisy, and so neural networks provide a method to smooth out this noise in a consistent way.

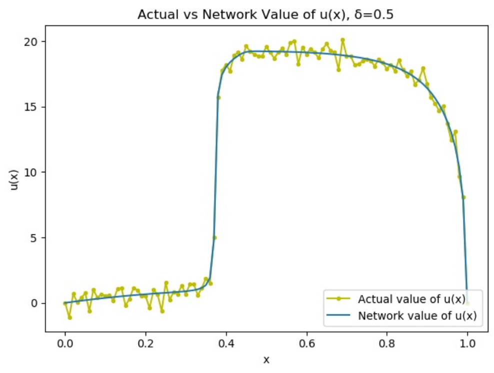

Figure 3.11: Neural network value of $u(x)$ when $\delta = .50$.

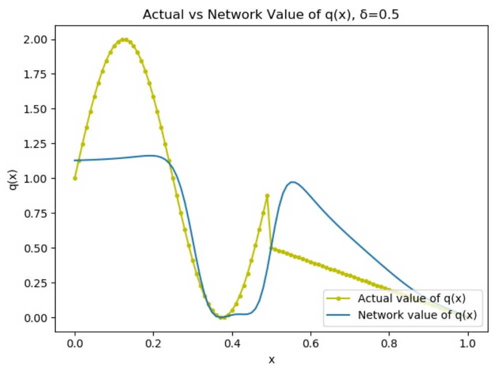

Figure 3.12: Neural network value of $q(x)$ when $\delta = .50$.

$q = \begin{cases} \sin(4\pi x)+1, & 0 \le x < 0.5 \\ -x+1, & 0.5 \le x \le 1 \end{cases}$								
$\delta = 0$					$\delta = 0.10$			
#it	Time	$\|\|u-\hat{u}\|\|$	$\|\|q-\hat{q}\|\|$		#it	Time	$\|\|u-\hat{u}\|\|$	$\|\|q-\hat{q}\|\|$
1215	106sec	5.53e-03	6.31e-02		1256	107sec	2.87e-02	1.56e-01
$\delta = 0.20$					$\delta = 0.30$			
#it	Time	$\|\|u-\hat{u}\|\|$	$\|\|q-\hat{q}\|\|$		#it	Time	$\|\|u-\hat{u}\|\|$	$\|\|q-\hat{q}\|\|$
1980	205sec	5.23e-02	1.48e-01		4309	712sec	8.19e-02	2.19e-01
$\delta = .40$					$\delta = 0.50$			
#it	Time	$\|\|u-\hat{u}\|\|$	$\|\|q-\hat{q}\|\|$		#it	Time	$\|\|u-\hat{u}\|\|$	$\|\|q-\hat{q}\|\|$
2829	348sec	1.10e-01	3.20e-01		3427	457sec	1.46e-01	3.69e-01

Table 3.5: Performance comparison between different noise values for the 1D Poisson equation with a discontinuous coefficient.

As table 3.5 indicates, as soon as you add noise to the measurements of q, the order of magnitude of the error for the network prediction increases. However, this new order of magnitude is consistent across all noise levels tested, which shows that the network still performs relatively well in all of the noisy cases. This example suggests that the idea of neural network augmentation can be reliably extended to more practical scenarios where the measurements of u are likely to be noisy.

All of the simulations in this section demonstrate that neural networks are an effective way of guessing irregular $q(x)$ coefficients for the inverse 1D Poisson problem. The most effective networks are those that estimate q using a noise-free u with enough parameters to accurately capture q. However, a noisy u and limited number of parameters do not significantly impact the ability of the network to perform well and could be considered acceptable in many practical applications.

3.2.3 Predicting $q(x)$ with Limited Observations

We now perform simulations where we assume that we only have access to measurements of $u(x)$ at a distance d from the boundary. In this situation, the error functional (2.3) that we want to minimize simplifies to

$$J(u(q)) = \frac{1}{2}\int_0^d |u_{meas} - u|^2 dx + \frac{1}{2}\int_{1-d}^1 |u_{meas} - u|^2 dx. \tag{3.16}$$

In [9], Berg and Nyström found that, even with values as small as $d = .1$, a neural network was able to reconstruct $q(x)$ in the entire domain. We reproduce this result here with $q(x) = 1$, $\delta = 0.05$, and $d = 0.1$. While figures 3.13 and 3.14 are not identical to those found in [9], the general pattern and error values are the same. The difference is likely due to the random initialization of the weights and noise values in the code as opposed to an implementation flaw, and so these results still validate the method we use in this thesis.

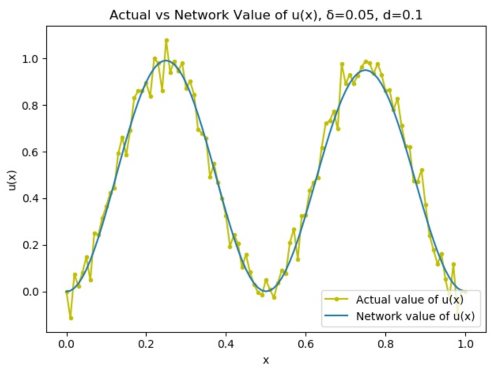

Figure 3.13: Neural network value of $u(x)$ when $q(x) = 1$, $\delta = 0.05$, and $d = 0.1$.

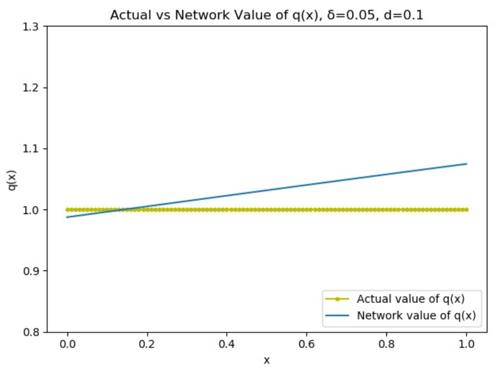

Figure 3.14: Neural network reconstruction of $q(x) = 1$ when $\delta = 0.05$ and $d = 0.1$.

$q(x) = 1$, $\delta = 0.05$, $d = 0.1$			
#it	Time	$\|\|u - \hat{u}\|\|$	$\|\|q - \hat{q}\|\|$
8	2sec	2.20e-02	4.01e-02

Table 3.6: Performance of the neural network for the 1D Poisson equation with a continuous coefficient and a noise value of $\delta = 0.05$ where $d = 0.1$.

The only $q(x)$ function presented in [9] is $q(x) = 1$. We now explore more complicated coefficients to see if the same results hold.

When $q(x) = 1 + x^2$, a network with a 9 nodes in the hidden layer is not able to correctly capture the behavior of $q(x)$ when $d = .47$. However, as soon as $d = .48$, the network is suddenly able to capture $q(x)$ correctly. This phenomenon is shown in figures 3.15-3.16 below.

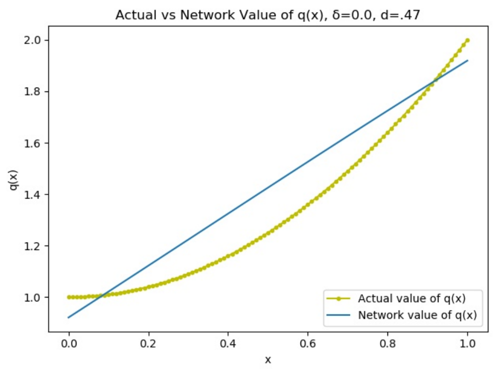

Figure 3.15: Predicted $q(x)$ when $d = .47$ and $q(x) = 1 + x^2$.

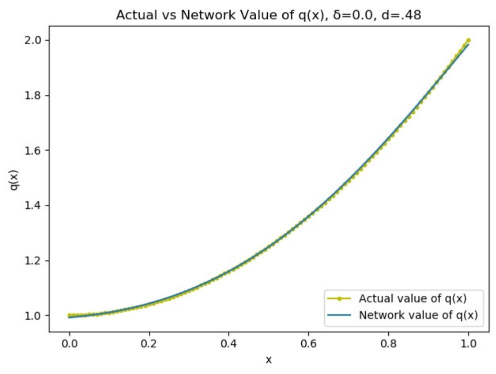

Figure 3.16: Predicted $q(x)$ when $d = .48$ and $q(x) = 1 + x^2$.

$q(x) = 1 + x^2$								
$d = .47$					$d = .48$			
#it	Time	$\|\|u - \hat{u}\|\|$	$\|\|q - \hat{q}\|\|$		#it	Time	$\|\|u - \hat{u}\|\|$	$\|\|q - \hat{q}\|\|$
43	4sec	3.65e-02	1.18e-01		194	13sec	3.46e-04	3.55e-03

Table 3.7: Performance comparison between $d = .47$ and $d = .48$ when $q(x) = 1 + x^2$.

When testing other coefficients, the same pattern emerges - a critical value of d is needed by the network in order to correctly reconstruct $q(x)$ (see figures 3.17-3.18 and table 3.8 for another test case when $q(x) = 1 + 0.5\sin(2\pi x)$). Practically, this means that the network needs a certain number of measurements of u in order to be able to reconstruct q. This critical value changes depending on the coefficient q. For example, when $q(x) = 1$, the critical value is $d = .07$, which explains why Berg and Nyström found that the network could successfully reconstruct this coefficient with a value of $d = .1$.

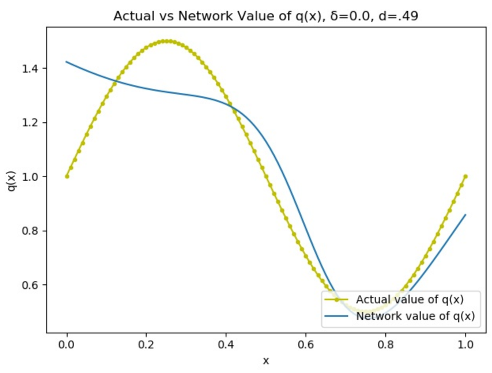

Figure 3.17: Predicted $q(x)$ when $d = .49$ and $q(x) = 1 + 0.5\sin(2\pi x)$.

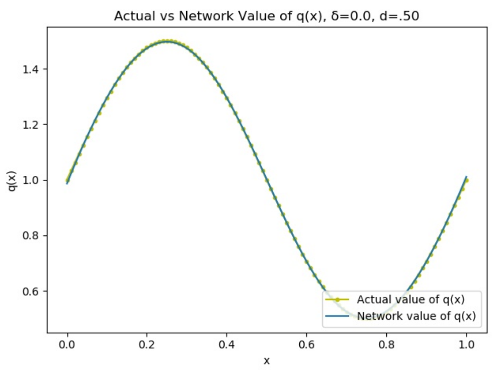

Figure 3.18: Predicted $q(x)$ when $d = .50$ and $q(x) = 1 + 0.5\sin(2\pi x)$.

$q(x) = 1 + 0.5\sin(2\pi x)$								
$d = .49$					$d = .50$			
#it	Time	$\|\|u - \hat{u}\|\|$	$\|\|q - \hat{q}\|\|$		#it	Time	$\|\|u - \hat{u}\|\|$	$\|\|q - \hat{q}\|\|$
142	9sec	1.66e-02	1.26e-01		838	58sec	1.76e-04	2.50e-03

Table 3.8: Performance comparison between $d = .49$ and $d = .50$ when $q(x) = 1 + 0.5\sin(2\pi x)$.

It is then plausible to ask whether the network architecture has a bearing on this critical value. As shown in tables 3.9-3.10 below, it seems that the network architecture - both in terms of the number of nodes and the number of layers - has little to no impact on this critical value.

	$q(x) = 1 + x^2$									
	1 Hidden Layer									
Num of Nodes	1	2	3	4	5	6	7	8	9	10
d	.46	.48	.48	.46	.48	.47	.48	.49	.48	.48
Num of Nodes	11	12	13	14	15	16	17	18	19	20
d	.48	.48	.49	.47	.48	.49	.48	.46	.48	.47
	2 Hidden Layers									
Num of Nodes	1	2	3	4	5	6	7	8	9	10
d	.47	.46	.47	.48	.46	.47	.47	.49	.48	.44
Num of Nodes	11	12	13	14	15	16	17	18	19	20
d	.49	.48	.47	.48	.20	.48	.49	.45	.48	.46

Table 3.9: Comparison of the data needed for different network architectures to accurately predict q when $q(x) = 1 + x^2$.

	$q(x) = 1 + 0.5\sin(2\pi x)$									
	1 Hidden Layer									
Num of Nodes	1	2	3	4	5	6	7	8	9	10
d	N/A	.50	.50	.48	.50	.50	.50	.50	.50	.48
Num of Nodes	11	12	13	14	15	16	17	18	19	20
d	.50	.50	.50	.50	.50	.50	.50	.50	.50	.50
	2 Hidden Layers									
Num of Nodes	1	2	3	4	5	6	7	8	9	10
d	N/A	.49	.49	.46	.49	.50	.47	.42	.47	.50
Num of Nodes	11	12	13	14	15	16	17	18	19	20
d	.50	.46	.41	.49	.50	.50	.50	.50	.47	.49

Table 3.10: Comparison of the data needed for different network architectures to accurately predict q when $q(x) = 1 + 0.5\sin(2\pi x)$.

In the tables above, the number of nodes corresponds to the number of nodes in the hidden layer, and for the network with 2 hidden layers, the number of nodes corresponds to the number of nodes in each of the hidden layers. As the tables show, the critical value of d changes little when compared to the network architecture, which suggests that neural networks cannot increase their probability of correctly guessing a q by scaling up their number of parameters if a sufficient amount of data about u is not first provided. As far as we are aware, this is the first time this particular phenomenon is being reported in the literature and so more research needs to be done to better understand this behavior.

The scenarios in this section demonstrate that some values of q need many observations of u in order to correctly reconstruct the coefficient, which shows that the amount of measurement data available to the user has a large impart of the performance of this method. All of these experiments also show that the relationship between the structure of the neural network and the network's ability to predict an unknown coefficient is not straightforward. However, constructing the best network for a given problem *a priori* is still a topic of much research.

3.3 Estimating the Diffusion Coefficient of the Poisson Equation for the Dimension $N = 2$

We now attempt to extend the ideas from section 3.2 to the 2D Poisson problem given by equation (3.1) above. For this problem, the functions u, f, and g are known and we would like to estimate the coefficient $q(x, y) : \mathbb{R}^2 \to (0, \infty)$. We let $\Omega = [0, 1] \times [0, 1]$ be the unit square which we discretize using a uniform mesh with 101×101 (i.e 10.201) elements and 20.000 vertices. The neural network we use for the 2D problem has 2 input layers, 2 hidden layers with 10 nodes, and 1 output layer (the 2 input nodes are for the x and y coordinates of the domain, respectively). With this network structure, 151 parameters characterize q and need to be optimized. The BFGS algorithm now stops iterating when the norm of the gradient is less than 10^{-7} and all of the parameters of the neural network are initialized with numpy's random number generator to a value in $[0, 1)$. For some tests, a noise value δr is added to each of the interior data points where $\delta \in [0, 1]$ and $r \in \mathcal{N}(0, 1)$ (the data on the boundary is always exact). Numpy's random seed generator is seeded with the value of 2 for reproducibility. Unless otherwise specified, all measurements of u in this section are noise-free and taken on the entire domain.

Berg and Nyström have demonstrated the ability of a neural network to correctly guess some simple q functions in this case [9]. In the sections below, we build on their idea by adding noise to the observations of u and by experimenting with more complex q functions, such as discontinuous q's. We also explore what happens when observations in the domain are limited.

3.3.1 Predicting an Oscillating q Function with Noise

In [9], Berg and Nyström demonstrated the ability of a neural network to recognize q in the scenario when $u = \sin(\pi x)\sin(\pi y)$ and $q = 1+0.5\sin(2\pi x)\sin(2\pi y)$. However, they did not analyze the scenario when the observations of u are noisy, and so we now provide these results below.

As in the 1D Poisson case, we add noise to the observations of u by letting δ increase in increments of .10 until $\delta = 0.50$. The results are summarized in table 3.11 below. Figure 3.20 shows the value of u in the most extreme case when $\delta = .50$. As you can see from figures 3.21-3.22, the neural network is still able to approximate the shape of the function q, even if the approximation is somewhat poor. This is still a nice result - it shows that even with a lot of noise, neural networks have a promising ability to smooth out the observations and reconstruct a continuous q that resembles the actual value of q.

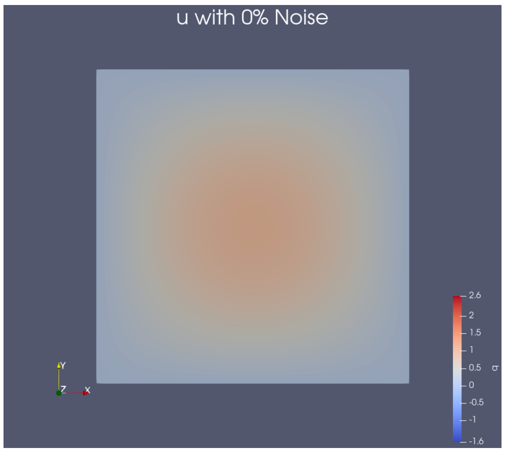

Figure 3.19: $u = \sin(\pi x)\sin(\pi y)$ when $\delta = 0$.

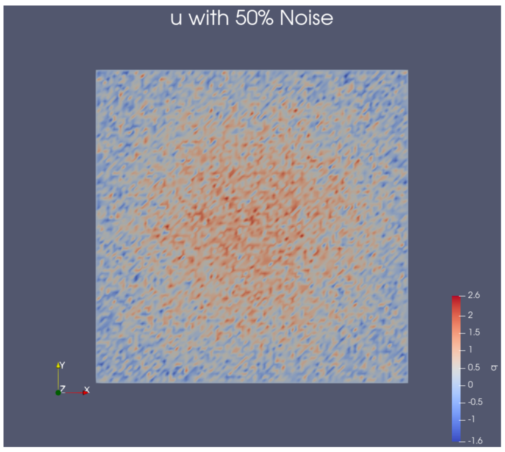

Figure 3.20: $u = \sin(\pi x)\sin(\pi y)$ when $\delta = 0.50$.

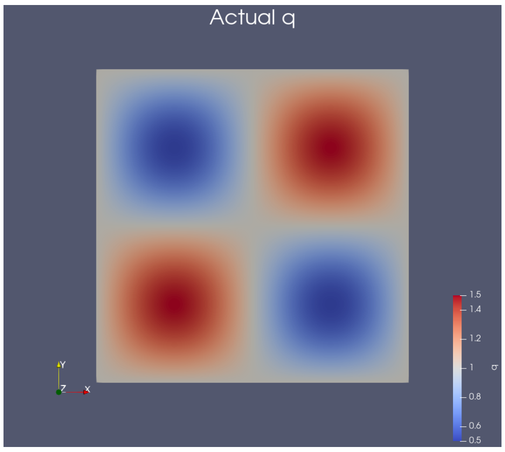

Figure 3.21: The q to be estimated by the network.

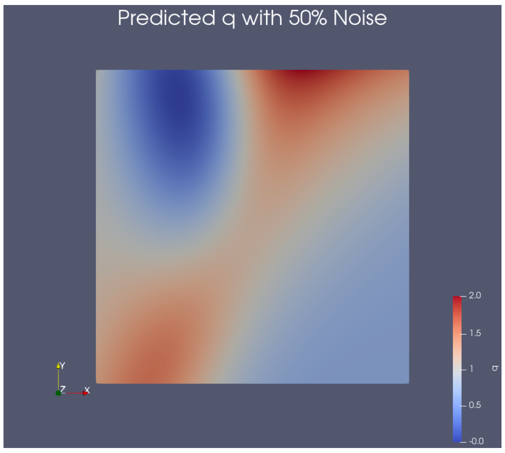

Figure 3.22: Predicted q estimated by the network when $\delta = .50$.

$q = 1 + 0.5\sin(2\pi x)\sin(2\pi y)$								
$\delta = 0$					$\delta = 0.10$			
#it	Time	$\|\|u - \hat{u}\|\|$	$\|\|q - \hat{q}\|\|$		#it	Time	$\|\|u - \hat{u}\|\|$	$\|\|q - \hat{q}\|\|$
3089	58min	9.48e-04	2.29e-01		10627	3hrs 40min	9.76e-03	3.99e-01
$\delta = 0.20$					$\delta = 0.30$			
#it	Time	$\|\|u - \hat{u}\|\|$	$\|\|q - \hat{q}\|\|$		#it	Time	$\|\|u - \hat{u}\|\|$	$\|\|q - \hat{q}\|\|$
3626	1hr 9min	1.31e-02	2.61e-01		3704	1hr 20min	2.79e-02	2.78e+00
$\delta = .40$					$\delta = 0.50$			
#it	Time	$\|\|u - \hat{u}\|\|$	$\|\|q - \hat{q}\|\|$		#it	Time	$\|\|u - \hat{u}\|\|$	$\|\|q - \hat{q}\|\|$
1384	29min	2.62e-02	3.63e-01		530	11min	2.52e-02	2.69e-01

Table 3.11: Performance comparison between different noise values for the 2D Poisson equation for an oscillating coefficient.

Notice that the time it takes for the algorithm to converge is significantly longer for the 2D Poisson problem than it is for the 1D Poisson problem. This is because the number of mesh elements jumps from 101 to 10.201, thereby greatly increasing the degrees of freedom present in the discretization of the PDE. This time is prohibitive for any sort of practical application. However, we adapt some ideas from [27] in chapter 5 to introduce an innovative method that reduces the computational complexity of this problem.

3.3.2 Predicting Discontinuous q Functions

We now extend the idea of neural network augmentation to discontinuous coefficients for 2D problems. This extension allows us to explore more unusual and interesting discontinuities that provide more insight into the effectiveness of the neural network predictions.

The first case we examine is a simple scenario where $f = 10$, $g = 0$, and q is given by the function

$$q = \begin{cases} .5, & [0, .5) \times [0, 1] \\ 1.5, & [.5, 1] \times [0, 1] \end{cases} \tag{3.17}$$

(see figure 3.23). The fact that $||u - \hat{u}|| = 2.13 \cdot 10^{-4}$ and $||q - \hat{q}|| = 3.44 \cdot 10^{-2}$ points to the ability of the neural network to successfully recognize jump discontinuities in the 2D Poisson case. This is further confirmed by figure 3.24.

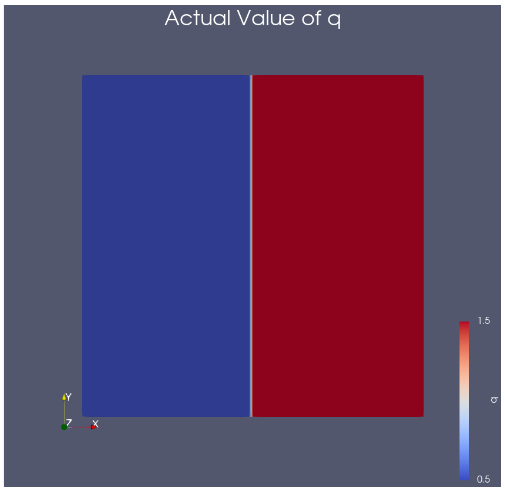

Figure 3.23: Actual discontinuous q to be estimated by the neural network - example 1.

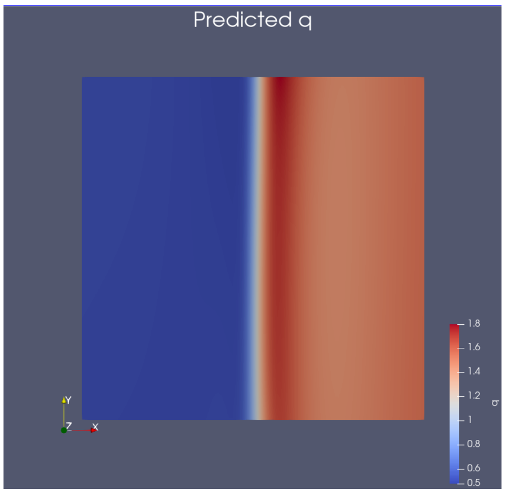

Figure 3.24: Predicted discontinuous q estimated by the neural network - example 1.

We now explore a more complicated scenario where $f = 10$, $g = 0$, and q is given by the function

$$q = \begin{cases} .5, & [0, .5) \times [.5, 1] \cup [.5, 1] \times [0, .5) \\ 1.5, & [0, .5) \times [0, .5) \cup [.5, 1] \times [.5, 1] \end{cases} \tag{3.18}$$

(see figure 3.25). In this scenario, $||u - \hat{u}|| = 8.68 \cdot 10^{-4}$ and $||q - \hat{q}|| = 4.29 \cdot 10^{-2}$, which points to good performance of the network. This is further confirmed by figure 3.26 below. This result is promising as it shows that neural networks are able to recognize more complicated discontinuous q functions in 2D provided the network has a sufficient number of parameters.

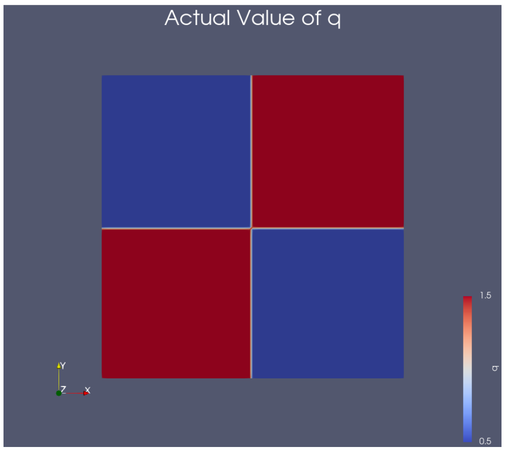

Figure 3.25: Actual discontinuous q to be estimated by the neural network - - example 2.

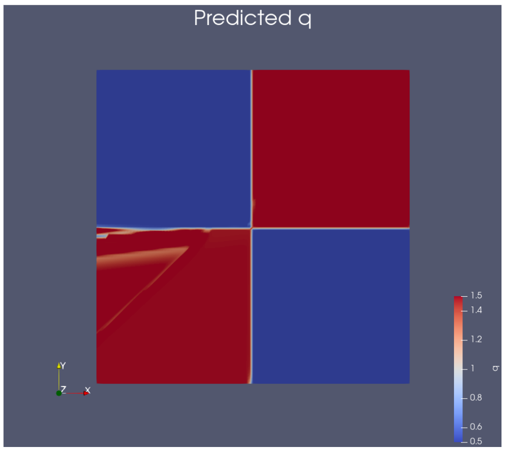

Figure 3.26: Predicted discontinuous q estimated by the neural network - example 2.

However, whenever q becomes too complicated, the network can no longer correctly identify it with a large amount of accuracy. For example, when

$$q = \begin{cases} .5, & [0, .33) \times [.33, .66) \cup [.33, .66) \times [0, .33) \cup \\ & [.33, .66) \times [.66, 1] \cup [.66, 1] \times [.33, .66) \\ 1.5, & \text{otherwise} \end{cases}, \tag{3.19}$$

the predictions of the network grow significantly larger, as evidenced by the error $||q - \hat{q}|| = 1.17$ (see figures 3.27-3.28). However, the network is still able to capture the general shape of the coefficient, which perhaps implies that the architecture of the network could be enriched in order to capture the discontinuities with the desired accuracy. However, the best method of designing a network for a given problem is still mostly heuristic because the set of functions a network can represent given a fixed architecture is still unknown.

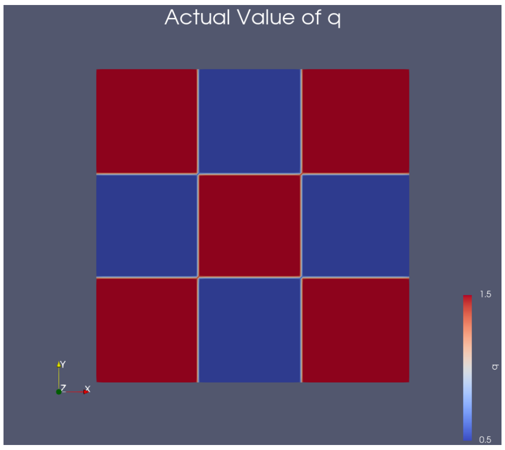

Figure 3.27: Actual discontinuous q to be estimated by the neural network - example 3.

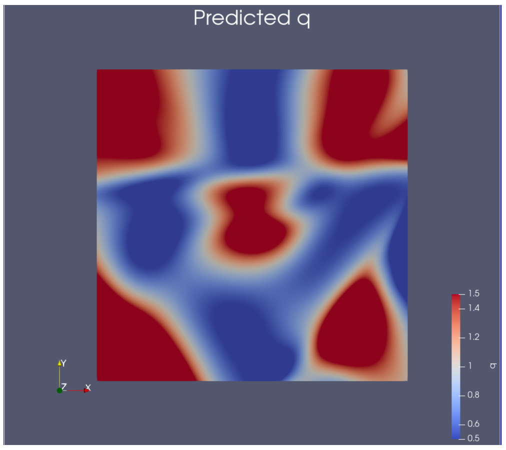

Figure 3.28: Predicted discontinuous q estimated by the neural network - example 3.

The results from this section are summarized in table 3.12 below.

$q = \begin{cases} .5, & [0,.5)\times[0,1] \\ 1.5, & [.5,1]\times[0,1] \end{cases}$			
#it	Time	$\lvert\lvert u-\hat{u}\rvert\rvert$	$\lvert\lvert q-\hat{q}\rvert\rvert$
2036	36min	2.13e-04	3.44e-02
$q = \begin{cases} .5, & [0,.5)\times[.5,1]\cup[.5,1]\times[0,.5) \\ 1.5, & [0,.5)\times[0,.5)\cup[.5,1]\times[.5,1] \end{cases}$			
#it	Time	$\lvert\lvert u-\hat{u}\rvert\rvert$	$\lvert\lvert q-\hat{q}\rvert\rvert$
1476	28min	8.68e-04	4.29e-02
$q = \begin{cases} .5, & [0,.33)\times[.33,.66)\cup[.33,.66)\times[0,.33)\cup \\ & [.33,.66)\times[.66,1]\cup[.66,1]\times[.33,.66) \\ 1.5, & \text{otherwise} \end{cases}$			
#it	Time	$\lvert\lvert u-\hat{u}\rvert\rvert$	$\lvert\lvert q-\hat{q}\rvert\rvert$
6260	1hr 57min	2.36e-02	1.17e+00

Table 3.12: Performance comparison of the discontinuous coefficients for the 2D Poisson equation.

Chapter 4

Identification of Material Properties for Nonlinear Elasticity Problems

Given the promising results in the previous chapter, we now consider a new problem by augmenting a more complicated PDE with a neural network. In this chapter, we consider the solution of inverse problems related to solid mechanics with the aim of identifying Young's modulus in the case of steady nonlinear elasticity equations characterized by a nearly-incompressible neo-Hookean constitutive law. This is a problem commonly found in structural mechanics when one wants to predict the nonlinear behavior of materials that undergo large deformations. We examine both the 2D and the 3D problem by testing a variety of coefficients, sometimes with noisy and limited observations of the solution. The results demonstrate that neural networks are successful at predicting Young's modulus for nonlinear mechanics problems, although the computational cost increases significantly as compared to the Poisson equation from chapter 3. This is a novel extension of the technique in [9] that has both practical and theoretical implications for the study of inverse problems.

4.1 Variational Formulation of the Nonlinear Elasticity Equation with a Hyperelastic Material

For the nonlinear elasticity equation with a hyperelastic material, we take a different approach than we did for the Poisson equation to arrive at the variational formulation of the problem. For this problem, we express the solution as the minimum of a suitable energy functional.

Given a domain $\Omega \subset \mathbb{R}^N$ where $N \in \{1, 2, 3\}$, the nonlinear elasticity problem is to find a displacement field $u : \Omega \to \mathbb{R}^N$ that minimizes the total potential energy Π given by

$$\Pi = \int_\Omega \psi(u)\ dx - \int_\Omega B \cdot u\ dx - \int_{\partial\Omega} T \cdot u\ ds \tag{4.1}$$

where ψ is the elastic stored energy, B is the body force per unit volume, and T is the traction force per unit area. The minimization problem is then expressed as: Find a $u^* \in V$ such that

$$u^* = \min_{u \in V} \Pi \tag{4.2}$$

where $V = \{\boldsymbol{v} \in [H^1(\Omega)]^N \mid \boldsymbol{v}|_{\Gamma_D} = \boldsymbol{0}\}$ and Γ_D is the Dirichlet part of the boundary.

To fully define the elastic stored energy ψ, we use the same model from [13] and consider the deformation gradient $F = I + \nabla u$ where I is the identity tensor. We can then define the right Cauchy-Green tensor as $C = F^T F$ and the scalars $J = \det(F)$ and $I_c = \mathrm{Tr}(C)$. The neo-Hookean stored energy model for ψ is then

$$\psi = \frac{\mu}{2}(I_c - 3) - \mu \ln(J) + \frac{\lambda}{2} \ln(J)^2$$

where μ and λ are the Lame parameters. In order to make a stronger parallel between the nonlinear and linear elasticity cases, the Lame parameters can be expressed in terms of two other coefficients E and ν where

$$\lambda = \frac{E\nu}{(1+\nu)(1-2\nu)}, \qquad \mu = \frac{E}{2(1+\nu)}.$$

As a slight abuse of language, we will refer to E as Young's modulus and ν as Poisson's ratio. For a more thorough introduction to nonlinear elasticity, see [11].

To arrive at the variational formulation of (4.2), let

$$L(u; v) = D_v \Pi = \left.\frac{d\Pi(u + \epsilon v)}{d\epsilon}\right|_{\epsilon=0}$$

be the directional derivative of Π with respect to u. Note that, at the minimum points of Π, $L(u; v) = 0 \ \forall \ v \in V$. The variational formulation of the problem is then: Find a $u \in V$ such that

$$L(u; v) = 0 \qquad \forall \ v \in V. \tag{4.3}$$

As in section 3.1, this variational formulation is discretized using linear continuous piecewise polynomials and transformed into an algebraic system. However, this time the variational form $L(u; v)$ is nonlinear in u, and so we use Newton's method with the Jacobian of L in order to solve the resulting system.

For the inverse problem, we will consider the situation where u is given and Young's modulus E needs to be estimated. To get the value of u needed for the inverse problem, we first solve the minimization problem with the value of E we want to predict. After u is obtained, we "forget" this value and the minimization problem (2.5) is solved in an attempt to "re-obtain" the correct value of E.

4.2 Estimating Young's Modulus in the Nonlinear Elasticity Equation for the Dimension $N = 2$

For the 2D problem, we consider a unit square domain $\Omega = [0, 1] \times [0, 1]$ with mixed boundary conditions

$$\Gamma_D = \{0\} \times [0, 1]$$
$$\Gamma_N = \partial\Omega \setminus \Gamma_D$$

where Γ_D is the Dirichlet boundary and Γ_N is the Neumann boundary. On both boundaries, $u(x) = (0, 0)$. Also, we let $B = (0, -1)$ and $\nu = 0.3$. This represents an object fixed to a wall that does not have any traction force applied to its boundary but does have a downward force acting upon it.

The neural network we use has 2 input layers, 1 hidden layer with 10 nodes, and 1 output layer. This network has one less layer than the network used for the 2D Poisson equation in section 3.3 since it was discovered that a network with a smaller number of parameters performed better for this problem. With this network structure, 41 parameters characterize E and need to be optimized. The BFGS algorithm stops iterating when the norm of the gradient is less than 10^{-7}. All of the parameters of the neural network are initialized with numpy's random number generator to a value between $[0, 1)$. For reproducible results, the seed of the random number generator is set to 2. Unless otherwise specified, all measurements of u in this section are noise-free and taken on the entire domain..

A uniform triangular mesh with 25×25 (i.e. 625) elements and 1152 vertices is used to discretize the domain (see figure 4.1).

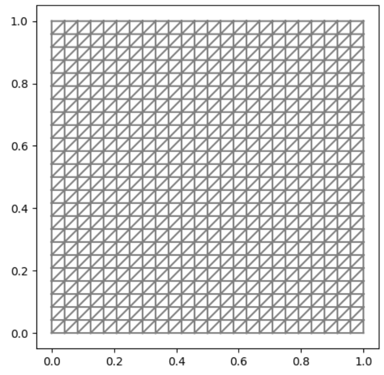

Figure 4.1: 2D mesh for the nonlinear elasticity problem.

4.2.1 Predicting Constant E Functions

The first problem we consider is trying to predict an E that is constant across the domain. We choose the value $E = 10$, which leads to the deformation shown in figure 4.3. As shown by figure 4.4, the neural network is able to predict this E with an accuracy that is on the order of 10^{-2}, which demonstrates the good predictive ability of the network in this scenario.

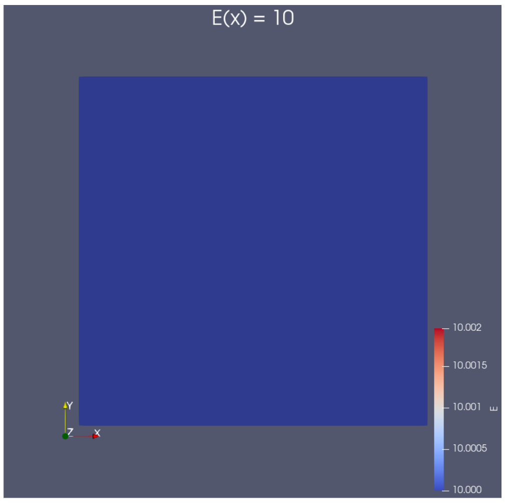

Figure 4.2: Actual value of $E(x)$.

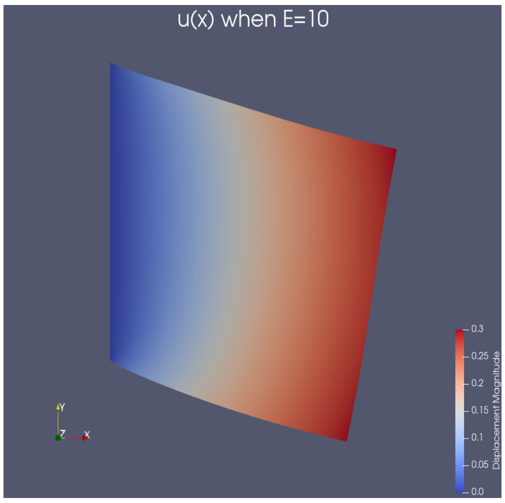

Figure 4.3: $u(x)$ when $E(x) = 10$.

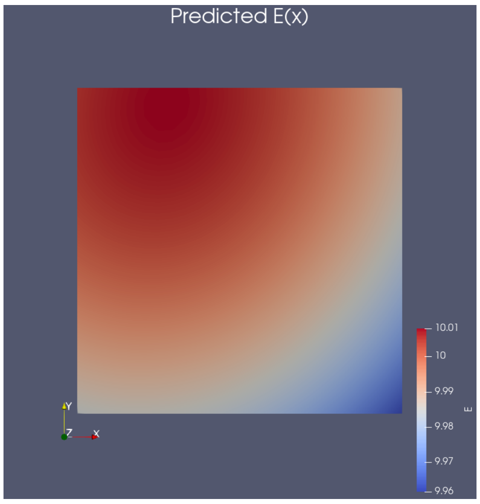

Figure 4.4: Neural network value of $E(x)$.

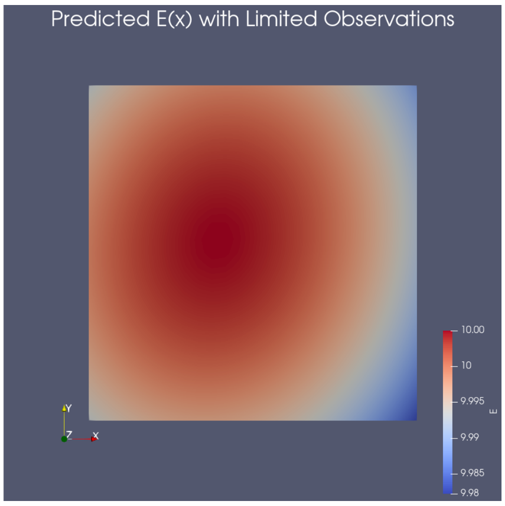

Figure 4.5: Neural network value of $E(x)$ with limited observations.

Following the methodology for the Poisson problem, we now limit the observations of u to the boundary of the domain, which is equivalent to minimizing the functional J over the boundary instead of over the whole domain. More specifically, we now take measurements of the displacement that are acquired on Γ_N. The results are shown in figure 4.5. Remarkably, these results show an improved performance of the neural network as compared to the case when all the data of u is available. While it is unclear exactly why this happens, part of the reason could be because the boundary gives the most information about the deformation of the object, and so aligning the boundary of the object to be within the threshold value for the termination of the BFGS algorithm actually aligns the whole object more accurately.

The results of these tests are summarized in table 4.1 below.

$E(x) = 10$								
All observations of u					Boundary observations of u			
#it	Time	$\|\|u - \hat{u}\|\|$	$\|\|E - \hat{E}\|\|$		#it	Time	$\|\|u - \hat{u}\|\|$	$\|\|E - \hat{E}\|\|$
55	29sec	3.07e-05	1.09e-02		56	27sec	8.49e-06	4.24e-03

Table 4.1: Performance comparison between having all the observations of u and just the boundary observations of u for the 2D nonlinear elasticity equation with a constant E.

Since the performance of the network is good when there is no noise, it makes sense to ask if the same results hold after noise is introduced. To test this scenario, we let δ increase in increments of .05 until $\delta = 0.25$ (this is different than the 1D Poisson case in section 3.2.2 as we discovered that the BFGS algorithm is unable to converge if u becomes too noisy). However, we only take observations of u on the boundary since this result performed better in the test case above. The results of adding noise to the observations of u are summarized in table 4.2 below.

$E(x) = 10$, Observations taken on the boundary of u								
$\delta = 0$					$\delta = 0.05$			
#it	Time	$\|\|u - \hat{u}\|\|$	$\|\|E - \hat{E}\|\|$		#it	Time	$\|\|u - \hat{u}\|\|$	$\|\|E - \hat{E}\|\|$
56	27sec	8.49e-06	4.24e-03		60	28sec	6.13e-03	2.27e-01
$\delta = 0.10$					$\delta = 0.15$			
#it	Time	$\|\|u - \hat{u}\|\|$	$\|\|E - \hat{E}\|\|$		#it	Time	$\|\|u - \hat{u}\|\|$	$\|\|E - \hat{E}\|\|$
60	27sec	1.23e-02	4.50e-01		56	27sec	1.84e-02	6.66e-01
$\delta = .20$					$\delta = 0.25$			
#it	Time	$\|\|u - \hat{u}\|\|$	$\|\|E - \hat{E}\|\|$		#it	Time	$\|\|u - \hat{u}\|\|$	$\|\|E - \hat{E}\|\|$
58	31sec	2.45e-02	8.83e-01		49	26sec	3.06e-02	1.09e+00

Table 4.2: Performance comparison between different noise values for the 2D nonlinear elasticity equation with a constant coefficient.

As the table shows, the ability of the network to correctly identify E weakens significantly as the noise value δ increases. As soon as any noise is added to u, the order of magnitude of the error of u increases by 3 and the order of magnitude of the error of E increases by 2. Figure 4.7 below shows the extreme case when $\delta = 0.25$. As shown from the figure, the prediction of E begins to look more linear as opposed to constant when noisy observations are taken from the boundary. This data shows that the 2D nonlinear elasticity equation presents more challenges to the neural network prediction of parameters than the 2D Poisson equation does.

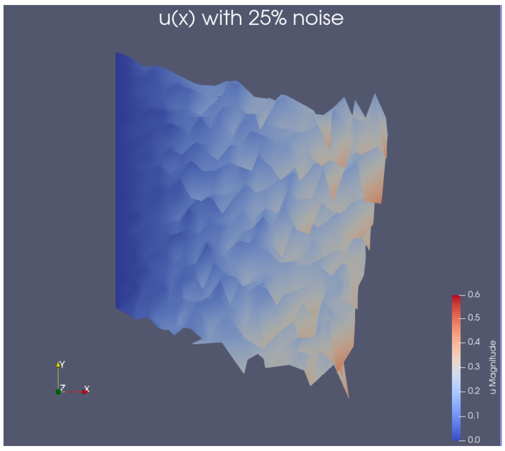

Figure 4.6: Value of $u(x)$ on the boundary with 25% noise.

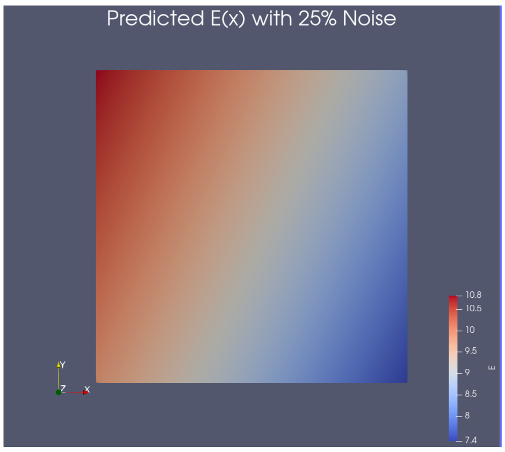

Figure 4.7: Neural network value of $E(x)$ when observing a u on the boundary with 25% noise.

4.2.2 Predicting Discontinuous E Functions

The next E we consider is one that has a sharp discontinuity in the middle of the domain. This simulates a simple situation in which an object is made from two distinct materials. More specifically, we let

$$E(x) = \begin{cases} 10, & x \in [0, 0.5) \times [0, 1] \\ 100, & x \in [0.5, 1] \times [0, 1] \end{cases}$$

This produces the $u(x)$ shown in figure 4.9. Since neural networks predict globally continuous functions, the discontinuity in E is harder for the network to capture. However, as shown in figure 4.10, the network is, in fact, relatively successful at capturing the discontinuity. However, the order of magnitude of the error for the value of E is 10, which is much greater than in the constant case. The error grows more when only the boundary of the domain is observed, as seen in figure 4.11 and table 4.3. This is also the opposite of what happens with a constant E, which shows that a discontinuous E presents more problems for the neural network than a constant E does.

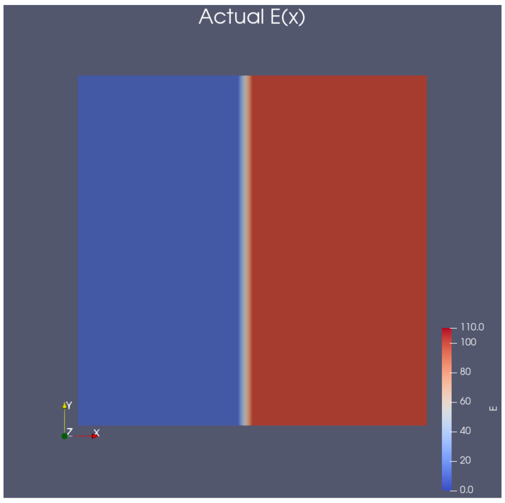

Figure 4.8: Actual value for a discontinuous $E(x)$ - example 1.

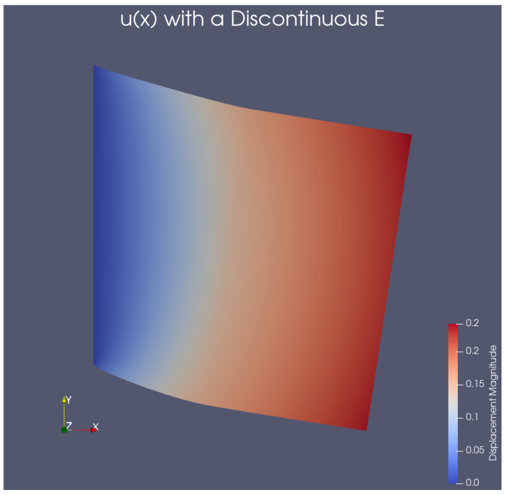

Figure 4.9: $u(x)$ with a discontinuous $E(x)$ - example 1.

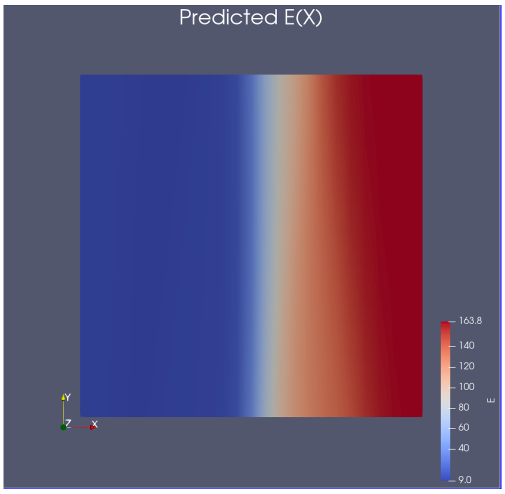

Figure 4.10: Neural network value for a discontinuous $E(x)$ - example 1.

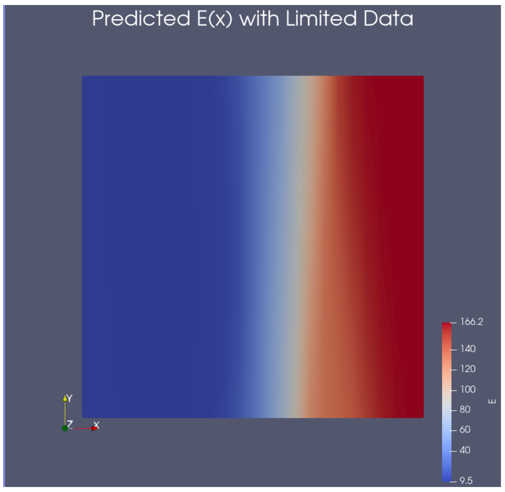

Figure 4.11: Neural network value for a discontinuous $E(x)$ with limited observations - example 1.

$E(x) = \begin{cases} 10, & x \in [0, 0.5) \times [0, 1] \\ 100, & x \in [0.5, 1] \times [0, 1] \end{cases}$								
All observations of u					Boundary observations of u			
#it	Time	$\|\|u - \hat{u}\|\|$	$\|\|E - \hat{E}\|\|$		#it	Time	$\|\|u - \hat{u}\|\|$	$\|\|E - \hat{E}\|\|$
172	51sec	6.58e-04	3.44e+01		218	109sec	6.08e-04	3.64e+01

Table 4.3: Performance comparison between having all the observations of u and just the boundary observations of u for the 2D nonlinear elasticity equation with a simple discontinuous coefficient.

The final discontinuous E we consider is one, like in section 3.3, that is broken into 4 uniform squares over the domain. More specifically,

$$E(x) = \begin{cases} 10, & [0, .5) \times [.5, 1] \cup [.5, 1] \times [0, .5) \\ 100, & [0, .5) \times [0, .5) \cup [.5, 1] \times [.5, 1] \end{cases}$$

as shown in figure 4.12. Figure 4.13 demonstrates that the neural network cannot recognize this particular E function. Instead, the network predicts an E that looks linear. The Netwon method used to solve the forward PDE problem is also unable to converge when observations are only taken on the boundary of the domain, as indicated by the "N/A" values in table 4.4.

It is unclear why the network fails in this case. It is possible that, because the inverse problem is not well-posed, more than one solution exists, and so the solution found by the network is plausible. It is also possible that the structure of the network does not permit it to guess this type of solution (although many other network structures were tested, none of which were able to recognize the correct E value). Another potential problem could be that the Newton method is initialized in such a way that the initial guess might be too far away from the solution, and so the method might not converge. In this scenario, reducing the number of optimization steps could potentially improve the results.

This scenario demonstrates that the nonlinear elasticity problem presents more difficulties for the neural network than the Poisson problem does, as the 2D Poisson problem is able to recognize a highly discontinuous coefficient pattern that resembles the pattern for E seen here. More experimentation needs to be done to determine why the network is unable to perform in this case.

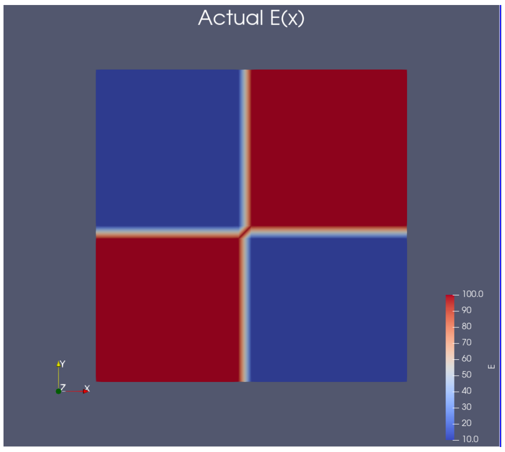

Figure 4.12: Actual value of $E(x)$ for a more complicated discontinuous coefficient.

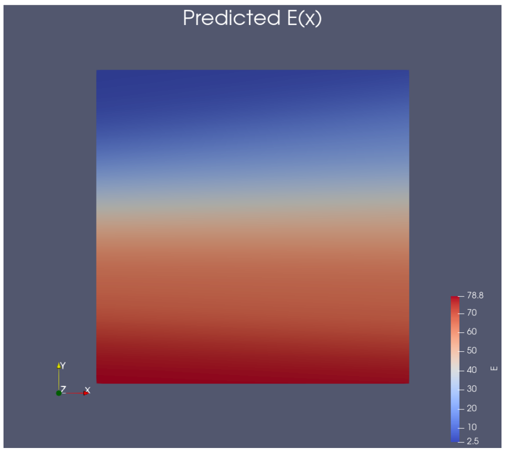

Figure 4.13: Neural network value of $E(x)$ for a more complicated discontinuous coefficient.

$E(x) = \begin{cases} 10, & [0,.5) \times [.5,1] \cup [.5,1] \times [0,.5) \\ 100, & [0,.5) \times [0,.5) \cup [.5,1] \times [.5,1] \end{cases}$								
All observations of u					Boundary observations of u			
#it	Time	$\|\|u-\hat{u}\|\|$	$\|\|E-\hat{E}\|\|$		#it	Time	$\|\|u-\hat{u}\|\|$	$\|\|E-\hat{E}\|\|$
140	28sec	3.62e-03	5.21e+01		N/A	N/A	N/A	N/A

Table 4.4: Performance comparison between having all the observations of u and just the boundary observations of u for the 2D nonlinear elasticity equation for a more complicated discontinuous coefficient.

Overall, the performance of the network in the discontinuous cases is less than ideal, and so no tests are performed where noise is added to the observations of u.

4.3 Estimating Young's Modulus in the Nonlinear Elasticity Equation for the Dimension $N = 3$

Many applications of nonlinear elasticity problems involve 3D objects, and so we now attempt to extend the results from the previous section to the 3D case. This time we consider a unit cube domain $\Omega = [0,1] \times [0,1] \times [0,1]$

with mixed boundary conditions

$$\Gamma_D = \{0\} \times [0,1] \times [0,1]$$
$$\Gamma_N = \partial\Omega \setminus \Gamma_D$$

where Γ_D is the Dirichlet boundary and Γ_N is the Neumann boundary. On both boundaries $u(x) = (0,0,0)$. Also, we let $B = (0,0,-1)$ and $\nu = 0.3$. This represents an object fixed to a wall that does not have any traction force applied to its boundary but does have a downward force acting upon it, as in the 2D scenario.

The neural network used to augment the problem has 3 inputs nodes, 1 hidden layer with 30 nodes, and 1 output node. The 3 input nodes are for the x, y, and z coordinates of the domain, respectively. With this network structure, 151 parameters characterize E and need to be optimized. The parameters of the network are initialized as described in the 2D problem above. Unless otherwise specified, all measurements of u in this section are noise-free and taken on the entire domain.

A uniform triangular mesh with $25 \times 17 \times 17$ (i.e. 7.225) elements and 36.864 vertices is used to discretize the domain (see figure 4.14).

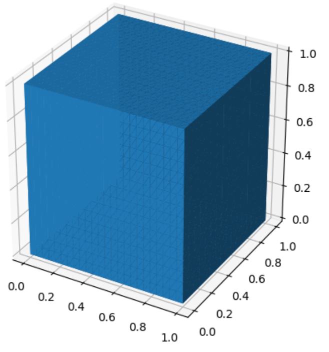

Figure 4.14: 3D mesh for the nonlinear elasticity problem.

4.3.1 Predicting Constant E Functions

We first consider an E that is constant across the domain. As in the 2D case, we start with the value $E = 10$. Figures 4.15 - 4.16 below show that the neural network can properly reconstruct this E value (see table 4.5 for the error and efficiency analysis).

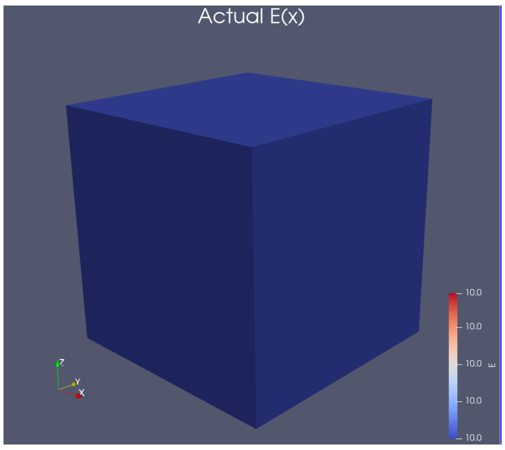

Figure 4.15: Actual value of $E(x)$ when $E(x) = 10$.

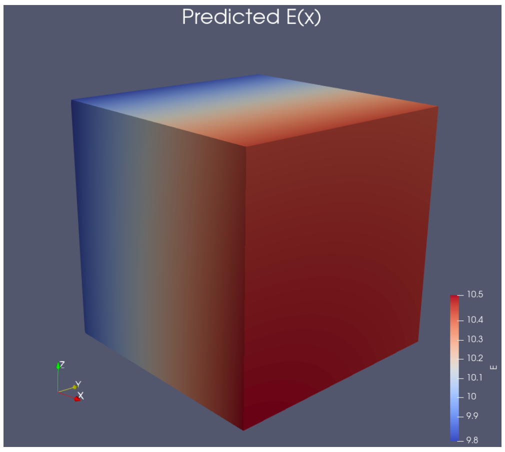

Figure 4.16: Neural network prediction of $E(x) = 10$.

As in the 2D case, we now limit the observations of u to the boundary. As shown in figure 4.17 below, this limitation does not affect the ability of the neural network to correctly identify the value of E. In fact, it actually improves the results as it did in the 2D case. For a more comprehensive analysis of these two cases, see table 4.5.

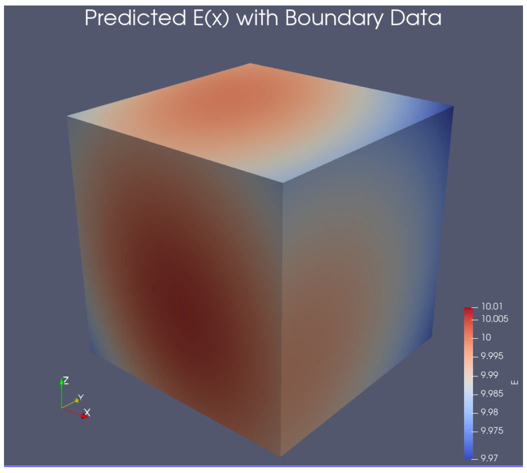

Figure 4.17: Neural network prediction of $E(x) = 10$ with data limited to the boundary.

$E(x) = 10$								
All observations of u					Boundary observations of u			
#it	Time	$\|\|u - \hat{u}\|\|$	$\|\|E - \hat{E}\|\|$		#it	Time	$\|\|u - \hat{u}\|\|$	$\|\|E - \hat{E}\|\|$
59	2hrs 17min	5.29e-04	2.64e-01		76	2hrs 41min	1.21e-05	6.26e-03

Table 4.5: Performance comparison between having all the observations of u and just the boundary observations of u for the 3D nonlinear elasticity equation for a constant coefficient.

Since we are now working in 3D, we can limit the observations on the boundary of the cube in more interesting ways to see if these limitations have any effect on the performance of the neural network. The first boundary we test is a checkerboard pattern on the boundary that incorporates 50% of the boundary data into the observations. More specifically, we let $\Omega = \Omega_0 \cup \Omega_1$ where

$$\begin{aligned}\Omega_0 = \{(x, y, z) \in \Omega : & \sin(8 * \pi * x) \sin(8 * \pi * x) \geq 0 \cup \\ & \sin(8 * \pi * x) \sin(8 * \pi * z) \geq 0 \cup \\ & \sin(8 * \pi * y) \sin(8 * \pi * z) \geq 0\}\end{aligned}$$

and $\Omega_1 = \Omega \setminus \Omega_0$. Observations are only taken on Ω_0 (see figure 4.18 below - the blue part of the cube is where we gather the observations).

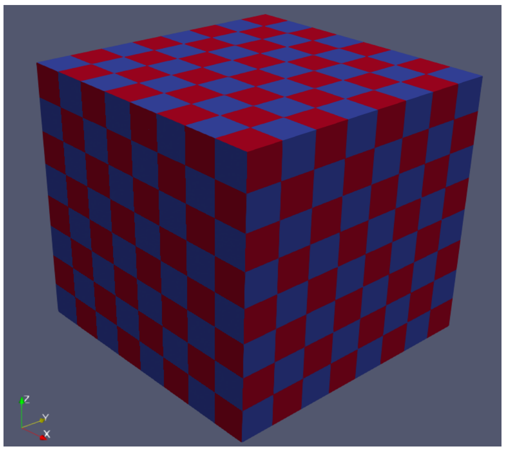

Figure 4.18: Checkerboard domain for the 3D nonlinear elasticity problem.

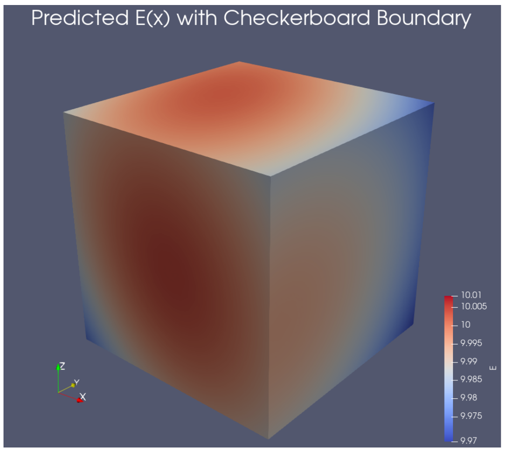

Figure 4.19: Neural network prediction of $E(x) = 10$.

As shown in figure 4.19, the checkerboard boundary has almost no effect on the ability of the neural network to accurately recognize the correct value of

E. This suggests that the network does not need a lot of data on the surface of the cube in order to correctly identify a simple E function.

Since the checkerboard case performed well, we decided to test further boundary restrictions to see if the same results hold. As in the 1D Poisson problem, we decide to take all observations within a certain distance d from the boundary. Figures 4.20-4.28 below show the various different boundaries tested (once again the blue part of the boundary is where observations are taken). The error and efficiency analysis for all of these cases is presented in table 4.6.

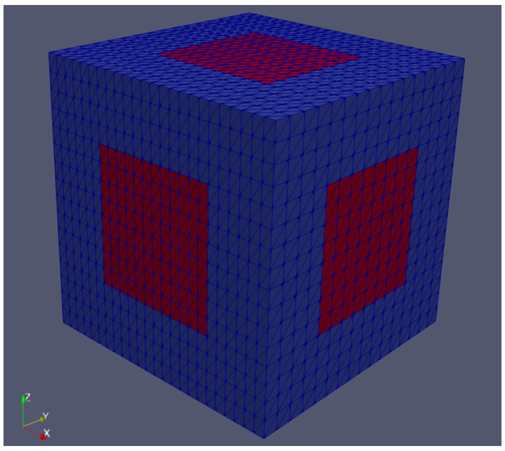

Figure 4.20: The unit cube with 50% of the observations taken from the boundary.

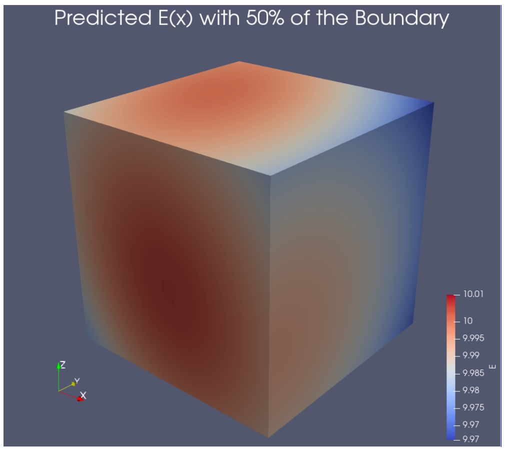

Figure 4.21: Neural network prediction of $E = 10$ with 50% of the observations taken from the boundary.

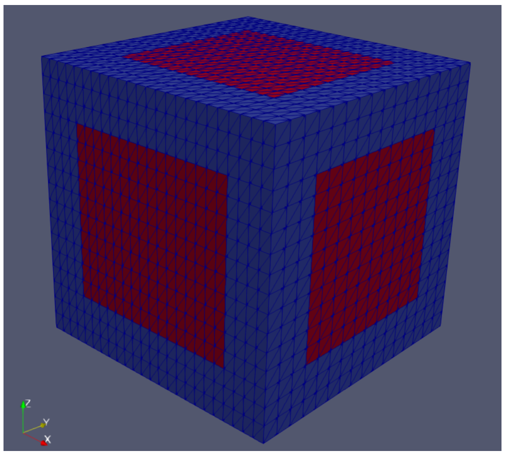

Figure 4.22: The unit cube with 30% of the observations taken from the boundary.

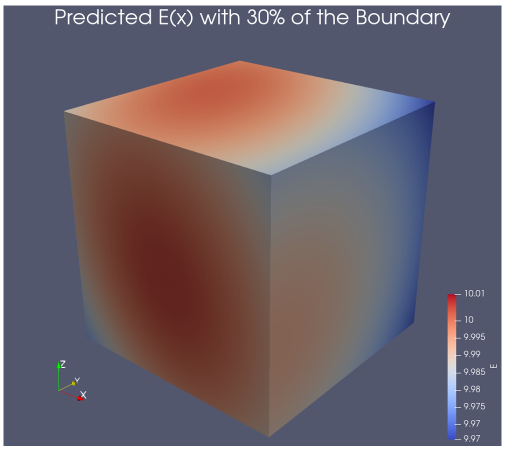

Figure 4.23: Neural network prediction of $E = 10$ with 30% of the observations taken from the boundary.

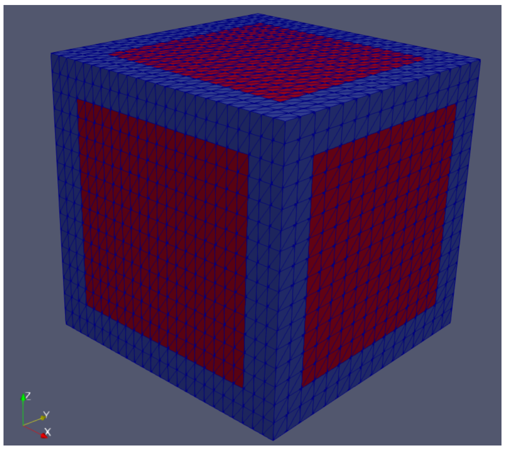

Figure 4.24: The unit cube with 20% of the observations taken from the boundary.

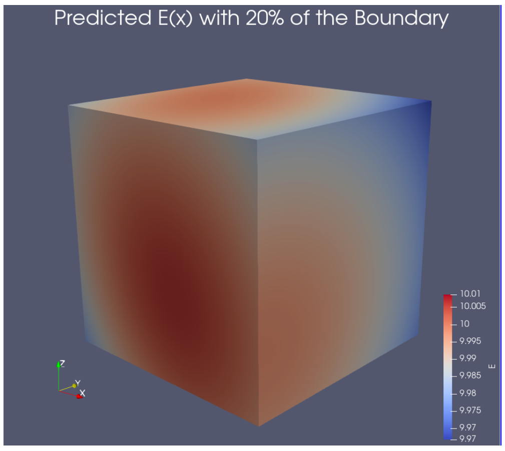

Figure 4.25: Neural network prediction of $E = 10$ with 20% of the observations taken from the boundary.

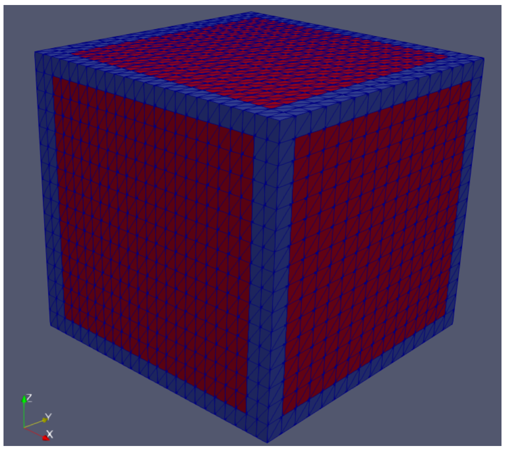

Figure 4.26: The unit cube with 10% of the observations taken from the boundary.

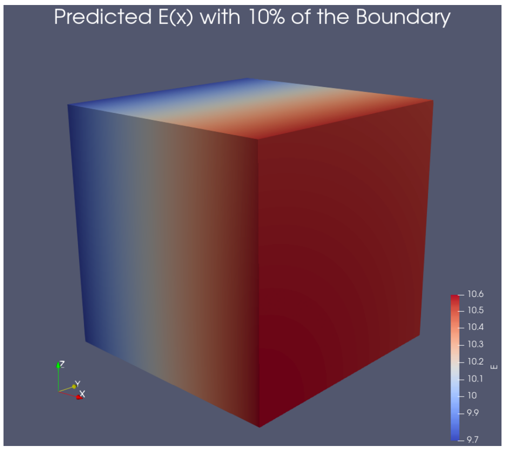

Figure 4.27: Neural network prediction of $E = 10$ with 10% of the observations taken from the boundary.

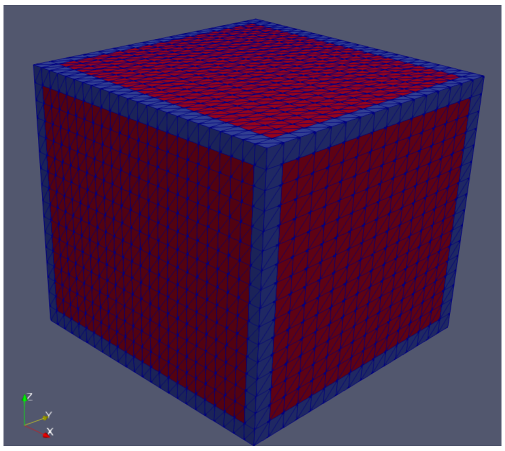

Figure 4.28: The unit cube with 5% of the observations taken from the boundary.

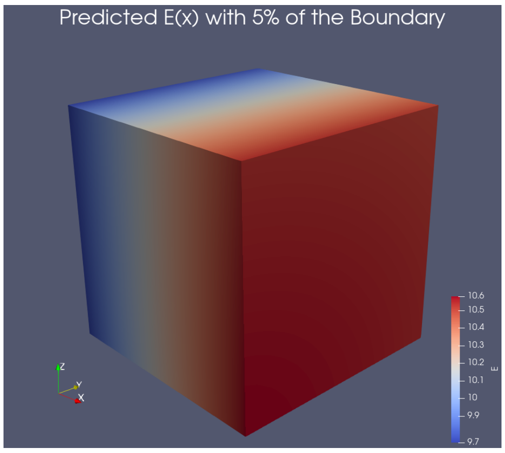

Figure 4.29: Neural network prediction of $E = 10$ with 5% of the observations taken from the boundary.

$E = 10$								
Checkerboard boundary with 50% data					50% of observations on the boundary			
#it	Time	$\|\|u - \hat{u}\|\|$	$\|\|E - \hat{E}\|\|$		#it	Time	$\|\|u - \hat{u}\|\|$	$\|\|E - \hat{E}\|\|$
82	4hrs 3min	1.59e-05	6.30e-03		82	3hrs 05min	1.25e-05	6.73e-03
30% of observations on the boundary					20% of observations the on boundary			
#it	Time	$\|\|u - \hat{u}\|\|$	$\|\|E - \hat{E}\|\|$		#it	Time	$\|\|u - \hat{u}\|\|$	$\|\|E - \hat{E}\|\|$
77	3hrs 18min	1.36e-05	6.93e-03		81	4hrs 08min	1.20e-05	6.15e-03
10% of observations on the boundary					5% of observations the on boundary			
#it	Time	$\|\|u - \hat{u}\|\|$	$\|\|E - \hat{E}\|\|$		#it	Time	$\|\|u - \hat{u}\|\|$	$\|\|E - \hat{E}\|\|$
59	2hrs 10min	7.20e-04	2.99e-01		59	2hrs 09min	6.85e-04	2.90e-01

Table 4.6: Performance comparison between different restrictions on the observations of u for the 3D nonlinear elasticity equation with a constant E.

As table 4.6 shows, the neural network is highly accurate at predicting the correct value of E until $\delta = .10$, at which point the error in E increases by 2 orders of magnitude. This seems to indicate that there is a sort of critical value - just like in the 1D Poisson case - where the network loses a large amount of its accuracy in estimating q. More research needs to be performed to determine if such a critical value exists and, if so, the primary factors that affect it.

4.3.2 Predicting Discontinuous E Functions

Following the pattern for the 2D nonlinear elasticity problem, we now explore what happens when trying to identify a discontinuous coefficient. More specifically, we let

$$E = \begin{cases} 10, & [0, .5) \times [0, 1] \times [0, 1] \\ 100, & [.5, 1] \times [0, 1] \times [0, 1] \end{cases}$$

as shown in figure 4.30. As figure 4.31 and table 4.7 demonstrate, the 3D case does seem to be able to recognize discontinuous E coefficients, although the order of magnitude of the error is fairly large. Also, due to the number of degrees of freedom of the mesh, these cases take a lot of computational power, and so they are more difficult to test efficiently (see chapter 5 below for a way to potentially speedup these computations). This initial test suggests that neural networks might be able to perform well for irregular coefficients in the 3D case, but more experimentation needs to be done with networks that have a higher number of parameters, more refined mesh structures, etc. in order to definitively determine the abilities and limitations of neural network augmentation in recognizing irregular coefficients for 3D domains.

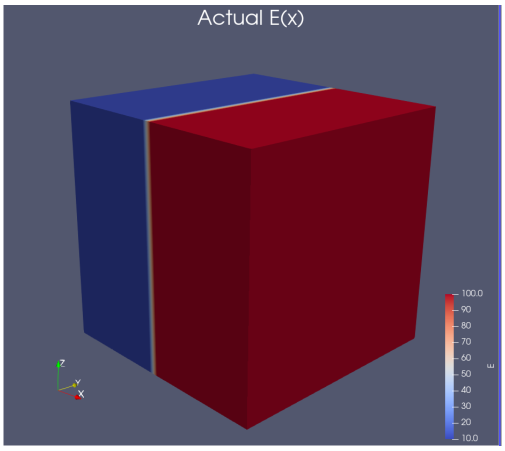

Figure 4.30: Actual discontinuous E to be predicted by the neural network.

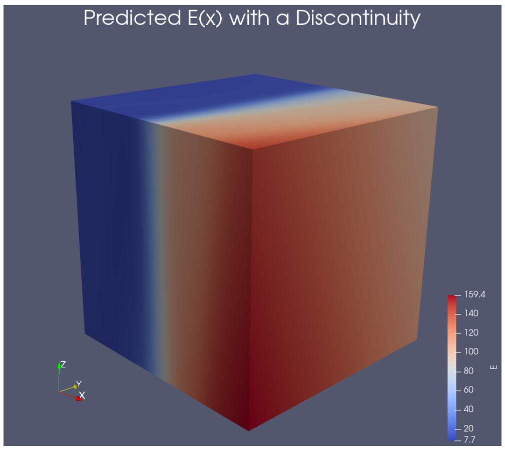

Figure 4.31: Discontinuous E predicted by the neural network.

$E = \begin{cases} 10, & [0,.5) \times [0,1] \times [0,1] \\ 100, & [.5,1] \times [0,1] \times [0,1] \end{cases}$			
#it	Time	$\|\|u - \hat{u}\|\|$	$\|\|E - \hat{E}\|\|$
344	13hrs 54min 46s	9.67e-04	2.09e+01

Table 4.7: Performance of the neural network for the 3D nonlinear elasticity equation with a discontinuous coefficient.

Chapter 5

Enhancing Computational Efficiency via the Reduced Basis Method

So far we have shown that neural networks can be effectively employed for the numerical solution of inverse problems for a wide variety of coefficients and differential equations. However, the results above also show that the method of solving the minimization problem (2.5) can be costly, particularly when the degrees of freedom of the mesh are high and the differential equation is more complex. This is due to the fact that the differential equation needs to be solved during every iteration of the BFGS algorithm. Since we are using the FEM on a fine mesh (i.e. we are using a high-fidelity model), this solve can be computationally expensive. To find a way to reduce the computational cost of solving the minimization problem (2.5), we want to focus on reducing the computational cost of this forward solve of the PDE.

One such method to reduce this cost is known as the reduced basis (RB) method. The initial design of RB methods began in the 1970's with the need for both the repetitive evaluation of a parameterized PDE problem and efficient parameter continuation methods for nonlinear problems [1, 14, 33]. These approaches were then extended to many different classes of differential equations, such as the Navier-Stokes equations in fluid dynamics [8, 19]. During the last decade, RB methods that utilize both greedy algorithms and proper orthogonal decomposition (POD) have been successfully used to speedup the computation of the solution to the forward parameterized PDE [10, 15, 24]. More recently, authors like [27] have been successful at combining RB methods with other techniques - such as a Bayesian approach to solve large inverse problems - to reduce the computational complexity of the original method.

The idea behind the reduced basis method is the assumption that the solution manifold for a PDE can be represented by a relatively small number of basis functions. A solve of the PDE then becomes equivalent to a Galerkin projection onto the space spanned by the reduced basis. While the cost of constructing the reduced basis space can be large, it only needs to be performed once per problem. Once the space has been constructed, the cost of the Galerkin projection onto the space is small. This means that reduced basis methods are a great tool to employ when a forward solve of a PDE problem needs to be evaluated multiple times, as is the case for our minimization problem.

In this chapter, we introduce a novel approach that combines the reduced basis method with the neural network augmentation of PDE problems to improve computational efficiency. We show that this approach has the potential to be effective in reducing computational cost while maintaining the accuracy of the prediction of the PDE coefficient. However, there are some theoretical and practical concerns that arise from the use of this method that are also outlined below. All of the simulations in this chapter are performed with the RBniCS package [17].

5.1 Introduction to the Reduced Basis Method

Since we are using a neural network to characterize the coefficient of a PDE, we can interpret the PDE system (2.1) from section 2.1 as a parametric problem. More specifically, we want to find a $u(\mu) \in V$ such that

$$\begin{aligned} F(u(\mu), q(\mu)) &= 0, \qquad \text{in } \Omega \\ u(\mu) &= g, \qquad \text{on } \partial\Omega \end{aligned} \tag{5.1}$$

where μ is a vector of parameters. We then define the solution manifold as the set that contains all of the solutions of the parametric equation (5.1) as μ varies between all possible values, i.e.

$$\mathcal{M} = \{u(\mu) : \mu \in \mathbb{P}\} \subset V$$

where $\mathbb{P}$ is the parameter domain and each $u(\mu) \in V$ is a solution to (5.1). We assume that this solution manifold has a low dimension, which is equivalent to saying that the span of a relatively low number of well-chosen basis functions can represent the manifold with a small error (note that this assumption may not be valid for the scenario presented in this thesis, see section 5.3). There are 2 parts to the RB method: constructing the RB space that approximates the solution manifold (known as the offline phase) and performing a Galerkin projection of a specific value of the parameter vector μ onto the RB space to approximate the forward solve of the PDE (known as the online phase).

During the offline phase, we seek to construct an N-dimensional subspace V_N of V by approximating the solution manifold $\mathcal{M}$ with an N-dimensional manifold $\mathcal{M}_N$. Many methods exist to construct this reduced basis space, such as proper orthogonal decomposition (POD) or a greedy algorithmic approach, the latter of which we utilize here. To use the greedy algorithmic approach, we first introduce a finite-dimensional set of points in the parameter domain $\mathbb{P}_h \subset \mathbb{P}$ where $\dim(\mathbb{P}_h) < \infty$. We can then generate the manifold $\mathcal{M}_h(\mathbb{P}_h) = \{u(\mu) : \mu \in \mathbb{P}_h\} \subset \mathcal{M}$. As long as $\mathbb{P}_h$ is fine enough, $\mathcal{M}_h$ is a good representation of $\mathcal{M}$. The greedy algorithm is an iterative procedure that adds a basis function to V_N at each iteration. The greedy algorithm picks this function by maximizing the estimated model order reduction error of the previous space spanned by the existing basis functions. It requires one solution of the PDE to be computed at each iteration for a total of N solutions that need to be calculated in order to construct the space. This means that the offline procedure could, in theory, be computationally expensive. This is offset by the fact that it only needs to be performed once per problem. The advantage of using the greedy algorithm instead of POD is that is allows for $\mathbb{P}_h$ to be more dense, which is highly desirable for our problem since the range of the parameters is large. For more information on how the greedy algorithm works, see [17].

After the RB space has been constructed, we move to the online part of the method where we compute a solution $u_N(\mu)$ in the RB space V_N through a Galerkin projection. Formally, this procedure is similar to the one described in section 2.3 for the Galerkin Finite Element method; the only difference is that we are now projecting onto the reduced basis space V_N instead of the discrete subspace V_h. In order to speed up the computation, we assume that all the bilinear forms associated to the weak form of the problem have an affine decomposition (i.e. they are able to be written in terms of a sum of coefficients that only depend on μ multiplied with a form that does not depend on μ). This is called the affine assumption and it does not always hold. When it does not hold - which is the case encountered in this thesis - an approximate bilinear form can be found that does have an affine decomposition using a technique known as the empirical interpolation method (EIM) [7]. Ideally, the computational cost of the online portion of the algorithm is only dependent upon N, which makes the online evaluation of the PDE computationally efficient. For a more comprehensive overview of the RB method, the greedy algorithm, and the EIM method, see the excellent introduction provided by [17].

In our scenario, the PDEs we consider are parameterized by the weights and biases of the neural network. This means that the solution manifold $\mathcal{M}$ is generated by all the possible neural network functions that could represent the desired coefficient given a fixed network architecture.

5.2 The Reduced Basis Method Applied to the Poisson Problem

To study the effectiveness of the reduced basis method when combined with the neural network augmentation of a PDE problem, we examine the case of the Poisson equation from section 3.3:

$$\begin{aligned} -\nabla \cdot (q(x)\nabla u(x)) &= f(x), \qquad x \in \Omega \\ u(x) &= g(x), \qquad x \in \partial\Omega \end{aligned}$$

where $f = 10$ and $g = 0$. As mentioned in the section above, the solution manifold $\mathcal{M}$ is formed by all the possible representations of q attainable with a fixed neural network architecture. This means that $\mathcal{M}$ can become large quickly, particularly as the number of network parameters increases. The implication is that it is unlikely that $\mathcal{M}$ has a low dimension when the number of parameters of the network is large. To try and mitigate this problem, we use a neural network with 1 hidden layer of 10 nodes instead of 2 hidden layers with 10 nodes each (as was done in section 3.3) to reduce the number of parameters from 151 to 41. To select the subspace P_h, we limit each parameter to the range $[-100, 100]$. This range is heuristically selected in such a way that it still allows the neural network to capture a wide variety of functions while attempting to limit the size of P_h for computational efficiency. As a result, the parameter space $\mathbb{P}_h$ is the cross product of the interval $[-100, 100]$ for all 41 parameters. We let the RB method train over a training set of 50 parameter points during the offline phase. The maximum number of basis functions N is set to 50 while the tolerance for the error over the training set is set at $1 \cdot 10^{-6}$. The EIM algorithm trains over a set of 60 parameter points and allows, at most, 100 terms. The tolerance for the EIM algorithm is also set at $1 \cdot 10^{-6}$.

For this particular problem, the offline training of the RB method takes almost 6 hours to complete. There are several reasons as to why the offline phase could be so computationally expensive: the parameter space P_h is quite large and needs to be explored by both the EIM method and when the construction of the RB space is taking place, at least 50 complete solves of the PDE need to be performed, and the physical memory of the machine running the code is too small to store all of the checkpoints needed for the calculations (the last factor is the major contributor to the poor offline performance as many calculations need to be read and written to the hard drive). However, this offline procedure only needs to be performed once for all of the test cases outlined below.

Running an error analysis for both the EIM method and the approximation V_{50} of V on a test set of 50 observations produces the results shown in table 5.1. These results show that the average of the errors over the 50 element test set is small, and so we can expect the approximations provided by the EIM method and V_N to be relatively good.

EIM Error			RB Error	
Avg Error	Avg Relative Error		Avg Error	Avg Relative Error
2.51e-13	1.62e-15		2.98e-13	1.25e-12

Table 5.1: Error analysis for the EIM method and RB approximation of V.

To demonstrate the feasibility of using the RB method for neural network augmented PDEs, we start by setting $q = 10$. Figures 5.1-5.2 show the comparison of the predictions when using the high-fidelity FEM forward solve and the RB low-fidelity forward solve. As table 5.2 confirms, the reduced basis method is able to predict q with a reasonable amount of accuracy and in slightly less time. While the order of magnitude of the error is greater with the reduced basis method, this is to be expected - the solution manifold $\mathcal{M}$ is only approximated by $\mathcal{M}_h$. If needed, it is possible to enrich the reduced basis space during the offline phase in order to try and close the gap between these errors.

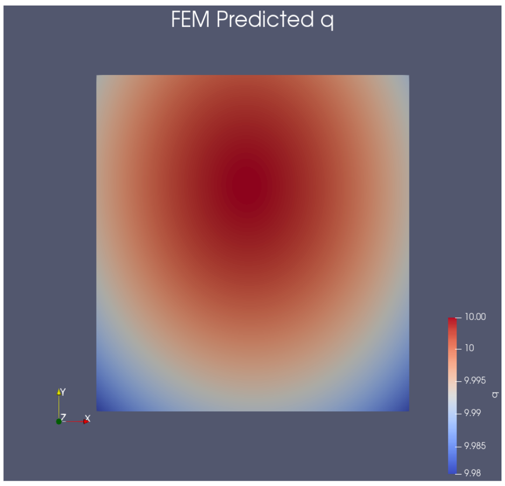

Figure 5.1: FEM prediction of q for a constant coefficient.

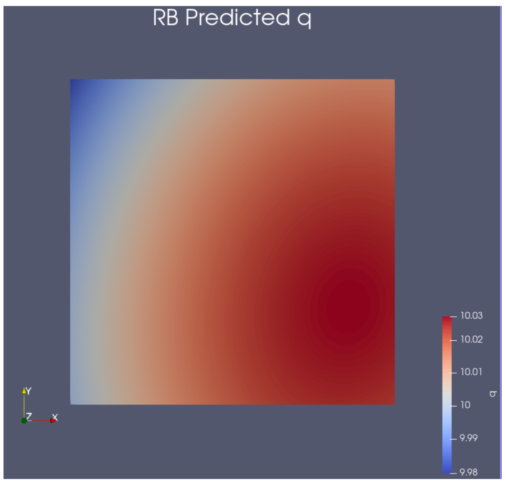

Figure 5.2: Reduced basis prediction of q for a constant coefficient.

$q = 10$								
FEM					Reduced Basis			
#it	Time	$\|\|u - \hat{u}\|\|$	$\|\|q - \hat{q}\|\|$		#it	Online Time	$\|\|u - \hat{u}\|\|$	$\|\|q - \hat{q}\|\|$
53	44sec	1.23e-05	5.26e-03		39	39sec	6.13e-05	1.80e-02

Table 5.2: Performance comparison between the FEM and RB methods of solving the Poisson problem for a constant coefficient.

Figures 5.3-5.4 and table 5.3 show that similar results hold for the scenario when q is linear in both x and y, e.g. $q = 1+x+y$. The improved performance provided by the RB method becomes significantly more evident in this case - over half of the computational cost is cut. This example demonstrates the potential speedup power of the RB method and highlights the clear benefits of this approach.

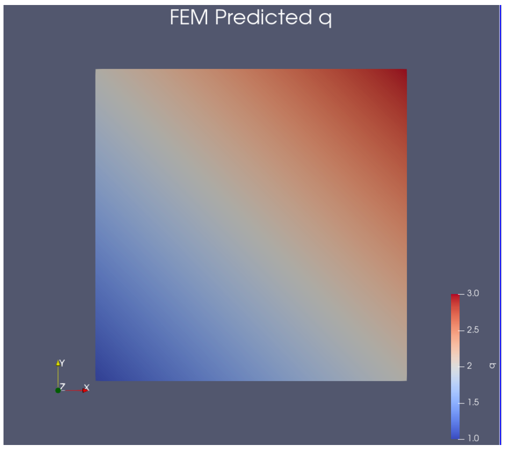

Figure 5.3: FEM prediction of q for a linear coefficient.

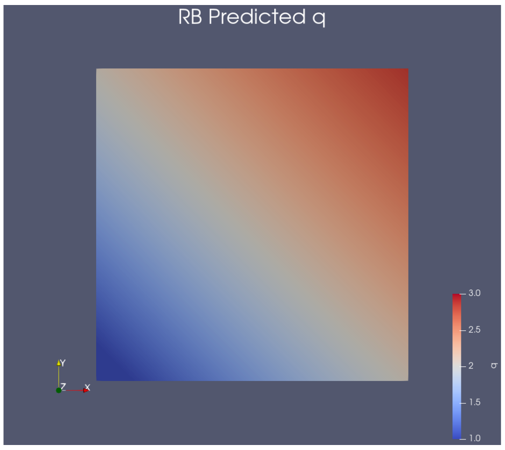

Figure 5.4: Reduced basis prediction of q for a linear coefficient.

$q = 1 + x + y$								
FEM					RB			
#it	Time	$\|\|u - \hat{u}\|\|$	$\|\|q - \hat{q}\|\|$		#it	Online Time	$\|\|u - \hat{u}\|\|$	$\|\|q - \hat{q}\|\|$
172	157sec	2.93e-04	4.11e-03		22	66sec	2.32e-03	4.39e-02

Table 5.3: Performance comparison between the FEM and RB methods of solving the Poisson problem for a linear coefficient.

However, the limitations of the RB method begin to become clear when examining a coefficient $q = 1 + x^2 + y^2$ that is quadratic in both x and y. As shown by figures 5.5-5.6, the RB method cannot calculate this q within a reasonable error bound. This is confirmed by table 5.4, which shows that, even though the speedup of the method is excellent, the order of magnitude of the error of q is 10^{-1}.

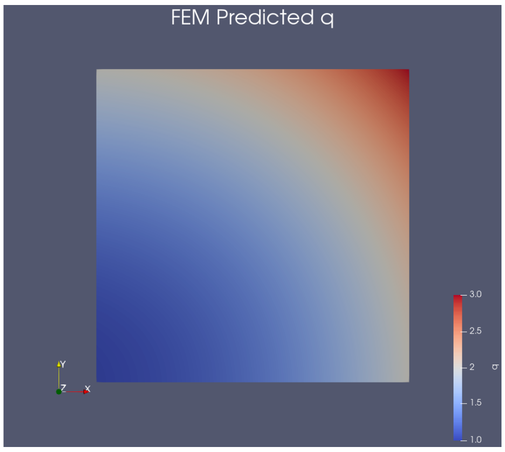

Figure 5.5: FEM prediction of q for a quadratic coefficient.

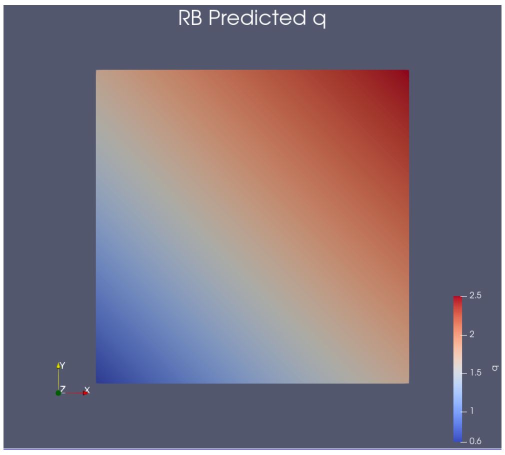

Figure 5.6: Reduced basis prediction of q for a quadratic coefficient.

$q = 1 + x^2 + y^2$								
FEM					RB			
#it	Time	$\|\|u - \hat{u}\|\|$	$\|\|q - \hat{q}\|\|$		#it	Online Time	$\|\|u - \hat{u}\|\|$	$\|\|q - \hat{q}\|\|$
848	800sec	2.71e-04	5.90e-03		22	84sec	7.28e-03	1.34e-01

Table 5.4: Performance comparison between the FEM and RB methods of solving the Poisson problem for a quadratic coefficient.

This phenomenon is further highlighted when we examine the coefficient

$$q = \begin{cases} .5, & [0, .5) \times [0, 1] \\ 1.5, & [.5, 1] \times [0, 1] \end{cases}$$

from section 3.3.2. Figures 5.7-5.8 show that, while the high-fidelity model is able to capture the discontinuity in the coefficient, the RB method smooths out the discontinuity across the domain, resulting in an increased error value for q. This increased error is seen in table 5.5. However, the shape of the discontinuity is, remarkably, still captured by the RB method. This observation is promising - while it means that a RB approach may not be able to fully capture q with the accuracy desired by the user, it can point the user towards the correct q value. This means that the RB approach could potentially be used to "precondition" the minimization problem by finding an appropriate initial guess for the parameters. This idea is explored more in section 5.3 below.

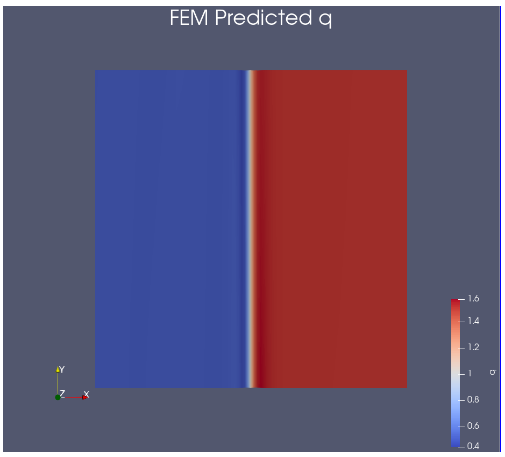

Figure 5.7: FEM prediction of q for a discontinuous coefficient.

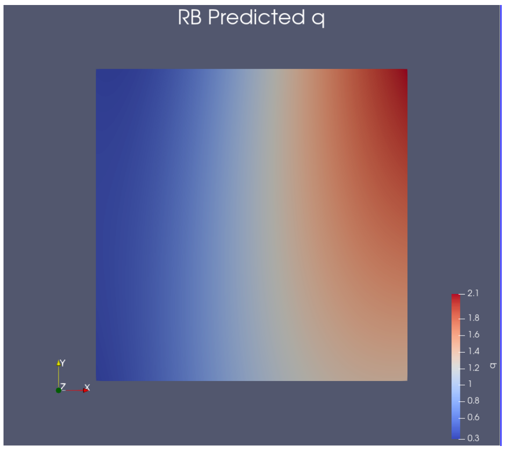

Figure 5.8: Reduced basis prediction of q for a discontinuous coefficient.

$q = \begin{cases} .5, & [0,.5) \times [0,1] \\ 1.5, & [.5,1] \times [0,1] \end{cases}$								
FEM					RB			
#it	Time	$\|\|u-\hat{u}\|\|$	$\|\|q-\hat{q}\|\|$		#it	Online Time	$\|\|u-\hat{u}\|\|$	$\|\|q-\hat{q}\|\|$
2360	2121sec	4.74e-04	3.57e-02		37	113sec	2.60e-02	2.40e-01

Table 5.5: Performance comparison between the FEM and RB methods of solving the Poisson problem for a discontinuous coefficient.

5.3 Open Issues and Challenges of the Reduced Basis Approach

As alluded to in the previous sections, the reduced basis method encounters some serious limitations when used to solve the forward problem for the PDE. The largest of these limitations is that one of the key assumptions of the method - namely that the solution manifold has a low dimension - is likely violated. This is due to the fact neural networks are characterized by a large number of parameters which increases the size of the parameter space $\mathbb{P}$. Further complicating the issue is the fact that the parameters of the network can, theoretically, be any real number. This also complicates the choice of $\mathbb{P}_h$. It is difficult to know *a priori* the range of acceptable parameters for the neural network in order to reconstruct the appropriate PDE coefficient, and so selecting $\mathbb{P}_h$ is nontrivial. These issues quickly make the RB method infeasible for any sort of neural network architecture that has a large number of parameters or for a network that estimates an overly-complicated q function.

However, the results in the preceding sections also indicate that the RB method could, perhaps, still be useful to speedup the computational cost of solving the minimization problem. We have shown that the RB method is capable of capturing the general structure of the coefficient. This means that the method could be used to produce an initial guess of the parameters for the minimization problem. Used in this way, the RB method acts as a sort of surrogate model for the minimization problem as it provides a starting point for the iterative algorithm that is already significantly closer to the end result as compared to a random starting point. This means that the algorithm would converge in less iterations, and therefore in less time. However, more research needs to be done to determine if such a method is feasible and to explore the potential benefits that may be gained by such an approach.

Chapter 6

Conclusion

In this thesis, we have extended the ideas of Berg and Nyström in [9] by showing how to augment two different PDEs with a neural network to predict various coefficients. While the test problems we have explored are relatively simple, the results demonstrate that neural networks are able to capture a large variety of coefficients under various conditions. The networks are also able to reconstruct these coefficients when the observations of the solution u are less than ideal (e.g. u contains noise or the number of observations of u is limited). We have also shown that the architecture of the network plays a role in determining how accurate these approximations are, although other factors - such as the number of observations of u - are more critical to the success of the method.

We have also introduced the novel idea of combining the reduced basis method with the neural network augmentation of the PDE in order to reduce the computational cost of solving the minimization problem. Specifically, we have shown that the reduced basis method is successful at reducing the computational time needed to perform different simulations while maintaining the accuracy of the prediction of q for relatively simple q's. This speedup is essential from a practical perspective. It shows that more complicated - and therefore computationally expensive - problems can be solved cheaply, which means that the inverse problem can be solved efficiently for a large range of PDE problems. While certain limitations need to be addressed and more complex applications need to be studied to determine the efficacy of this method, this thesis establishes a foundation from which to begin these explorations and demonstrates that such explorations are likely to be both feasible and practical in many different scenarios.

More research is needed to determine what coefficients a neural network can represent with a fixed architecture and what effect changing the architecture of the network has on the ability of the network to successfully recognize different coefficients. Also, the idea of using the reduced basis method as a low-fidelity solver in order to produce an educated guess for the numerical optimization procedure involving the high-fidelity finite element solver should be explored.

Bibliography

[1] B. Almroth, P. Stern, and F. Brogan. Automatic choice of global shape functions in structural analysis. 1978.

[2] Martin S. Alnæs, Jan Blechta, Johan Hake, August Johansson, Benjamin Kehlet, Anders Logg, Chris Richardson, Johannes Ring, Marie E. Rognes, and Garth N. Wells. The fenics project version 1.5. *Archive of Numerical Software*, 3(100), 2015.

[3] James A Anderson. *An introduction to neural networks*. MIT press, 1995.

[4] Pierre Argoul. Overview of Inverse Problems. Lecture, September 2012.

[5] Utkarsh Ayachit. *The ParaView Guide: A Parallel Visualization Application*. Kitware, Inc., Clifton Park, NY, USA, 2015.

[6] Claudine Badue, Rânik Guidolini, Raphael Vivacqua Carneiro, Pedro Azevedo, Vinicius Brito Cardoso, Avelino Forechi, Luan Jesus, Rodrigo Berriel, Thiago Meireles Paixão, Filipe Mutz, et al. Self-driving cars: A survey. *Expert Systems with Applications*, page 113816, 2020.

[7] Maxime Barrault, Yvon Maday, Ngoc Cuong Nguyen, and Anthony T Patera. An 'empirical interpolation'method: application to efficient reduced-basis discretization of partial differential equations. *Comptes Rendus Mathematique*, 339(9):667–672, 2004.

[8] A Barrett and G Reddien. On the reduced basis method. *ZAMM-Journal of Applied Mathematics and Mechanics/Zeitschrift für Angewandte Mathematik und Mechanik*, 75(7):543–549, 1995.

[9] Jens Berg and Kaj Nyström. Neural network augmented inverse problems for pdes, 2017.

[10] Lorenz Biegler, George Biros, Omar Ghattas, Matthias Heinkenschloss, David Keyes, Bani Mallick, Luis Tenorio, Bart van Bloemen Waanders, Karen Willcox, and Youssef Marzouk. *Large-scale inverse problems and quantification of uncertainty*. John Wiley & Sons, 2011.

[11] Philippe G Ciarlet. *Mathematical Elasticity: Volume I: three-dimensional elasticity*. North-Holland, 1988.

[12] B. Farley and W. Clark. Simulation of self-organizing systems by digital computer. *Transactions of the IRE Professional Group on Information Theory*, 4(4):76–84, 1954.

[13] Patrick E Farrell, Casper HL Beentjes, and Ásgeir Birkisson. The computation of disconnected bifurcation diagrams. *arXiv preprint arXiv:1603.00809*, 2016.

[14] RL Fox and H MlURA. An approximate analysis technique for design calculations. *AIAA Journal*, 9(1):177–179, 1971.

[15] David Galbally. *Nonlinear model reduction for uncertainty quantification in large-scale inverse problems: application to nonlinear convection-diffusion-reaction equation*. PhD thesis, Massachusetts Institute of Technology, 2008.

[16] Kevin Gurney. *An introduction to neural networks*. CRC press, 1997.

[17] Jan Hesthaven, Gianluigi Rozza, and Benjamin Stamm. *Certified Reduced Basis Methods for Parametrized Partial Differential Equations*. 01 2016.

[18] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks. *Neural Networks*, 3(5):551 – 560, 1990.

[19] Kazufumi Ito and Sivaguru S Ravindran. A reduced-order method for simulation and control of fluid flows. *Journal of computational physics*, 143(2):403–425, 1998.

[20] Victor Ivrii. 100 years of weyl's law. *Bulletin of Mathematical Sciences*, 6(3):379–452, 2016.

[21] S. Kabanikhin, N Tikhonov, V Ivanov, and M Lavrentiev. Definitions and examples of inverse and ill-posed problems. *Journal of Inverse and Ill-Posed Problems*, 16:317–357, 01 2008.

[22] Mark Kac. Can one hear the shape of a drum? *The American Mathematical Monthly*, 73(4):1–23, 1966.

[23] Ben Kröse, Ben Krose, Patrick van der Smagt, and Patrick Smagt. An introduction to neural networks. 1993.

[24] Toni Lassila, Andrea Manzoni, Alfio Quarteroni, and Gianluigi Rozza. A reduced computational and geometrical framework for inverse problems in hemodynamics. *International journal for numerical methods in biomedical engineering*, 29(7):741–776, 2013.

[25] Kenneth Levenberg. A method for the solution of certain non-linear problems in least squares. *Quarterly of Applied Mathematics*, 2(2):164–168, 1944.

[26] Stan Z. Li and Anil K. Jain. *Handbook of Face Recognition.* Springer Publishing Company, 2nd edition, 2011.

[27] Andrea Manzoni, Stefano Pagani, and Toni Lassila. Accurate solution of bayesian inverse uncertainty quantification problems combining reduced basis methods and reduction error models. *SIAM/ASA Journal on Uncertainty Quantification*, 4:380–412, 04 2016.

[28] Donald W. Marquardt. An algorithm for least-squares estimation of nonlinear parameters. *Journal of the Society for Industrial and Applied Mathematics*, 11(2):431–441, 1963.

[29] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[30] Joyce R. McLaughlin. *Overview of Inverse Problems*, pages 1119–1128. Springer Berlin Heidelberg, Berlin, Heidelberg, 2015.

[31] M.L. Minsky and S. Papert. *Perceptrons; an Introduction to Computational Geometry.* MIT Press, 1969.

[32] Sebastian K. Mitusch, Simon W. Funke, and Jørgen S. Dokken. dolfin-adjoint 2018.1: automated adjoints for fenics and firedrake. *Journal of Open Source Software*, 4(38):1292, 2019.

[33] Ahmed K. Noor. Recent advances in reduction methods for nonlinear problems. *Computers Structures*, 13(1):31 – 44, 1981.

[34] Travis E Oliphant. *A guide to NumPy*, volume 1. Trelgol Publishing USA, 2006.

[35] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[36] Alfio Quarteroni. *Numerical Models for Differential Problems.* Springer Publishing Company, 2nd edition, 2013.

[37] N. Rochester, J. Holland, L. Haibt, and W. Duda. Tests on a cell assembly theory of the action of the brain, using a large digital computer. *IRE Transactions on Information Theory*, 2(3):80–93, 1956.

[38] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, pages 65–386, 1958.

[39] Sandro Salsa. *Partial Differential Equations in Action: From Modelling to Theory).* Springer Publishing Company, 2nd edition, 2015.

[40] A. N. Tikhonov and V. IA. Arsenin. *Solutions of ill-posed problems / Andrey N. Tikhonov and Vasiliy Y. Arsenin ; translation editor, Fritz John.* Winston, 1977.

[41] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual.* CreateSpace, Scotts Valley, CA, 2009.

[42] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, CJ Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake Vand erPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul

van Mulbregt, and SciPy 1. 0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.

[43] Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

Acknowledgments

I am deeply grateful to Professor Andrea Manzoni (Politecnico di Milano) whose guidance, insight, and patience made this thesis possible. I would also like to thank those family members and friends whose countless hours of advice and encouragement pushed me to achieve my goals. None of this is possible without you.