Approximating the Electrical Impedance Tomography Forward Problem with Graph Neural Networks

Aleksi Mustonen

Thursday 3rd June, 2021

HELSINGIN YLIOPISTO — HELSINGFORS UNIVERSITET — UNIVERSITY OF HELSINKI

Tiedekunta/Osasto — Fakultet/Sektion — Faculty	Laitos — Institution — Department
Faculty of Science	Department of Mathematics and Statistics

Tekijä — Författare — Author
Aleksi Mustonen

Työn nimi — Arbetets titel — Title
Approximating the Electrical Impedance Tomography Forward Problem with Graph Neural Networks

Oppiaine — Läroämne — Subject
Applied mathematics

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
Master's thesis	June, 2021	41

Tiivistelmä — Referat — Abstract

Electrical impedance tomography is a differential tomography method where current is injected into a domain and its interior distribution of electrical properties are inferred from measurements of electric potential around the boundary of the domain. Within the context of this imaging method the forward problem describes a situation where we are trying to deduce voltage measurements on a boundary of a domain given the conductivity distribution of the interior and current injected into the domain through the boundary.

Traditionally the problem has been solved either analytically or by using numerical methods like the finite element method. Analytical solutions have the benefit that they are efficient, but at the same time have limited practical use as solutions exist only for a small number of idealized geometries. In contrast, while numerical methods provide a way to represent arbitrary geometries, they are computationally more demanding.

Many proposed applications for electrical impedance tomography rely on the method's ability to construct images quickly which in turn requires efficient reconstruction algorithms. While existing methods can achieve near real time speeds, exploring and expanding ways of solving the problem even more efficiently, possibly overcoming weaknesses of previous methods, can allow for more practical uses for the method.

Graph neural networks provide a computationally efficient way of approximating partial differential equations that is accurate, mesh invariant and can be applied to arbitrary geometries.
Due to these properties neural network solutions show promise as alternative methods of solving problems related to electrical impedance tomography.

In this thesis we discuss the mathematical foundation of graph neural network approximations of solutions to the electrical impedance tomography forward problem and demonstrate through experiments that these networks are indeed capable of such approximations. We also highlight some beneficial properties of graph neural network solutions as our network is able to converge to an arguably general solution with only a relatively small training data set. Using only 200 samples with constant conductivity distributions, the network is able to approximate voltage distributions of meshes with spherical inclusions.

Avainsanat — Nyckelord — Keywords
Deep Neural Networks, Electrical Impedance Tomography, Graph Kernel Networks

Säilytyspaikka — Förvaringsställe — Where deposited
Kumpula Campus Library

Muita tietoja — Övriga uppgifter — Additional information

Contents

Chapter 1

Introduction

In this thesis, we examine the electrical impedance tomography forward problem and approximation of its solutions using artificial neural networks, particularly graph neural networks.

Electrical impedance tomography is a differential tomography method where current is injected into a domain and the distribution of electrical properties, namely conductivity and permittivity, inside it are inferred from measurements of electric potential around the boundary of the domain. This imaging technique represents the inverse problem, constructing an image of the internal properties when boundary measurements are known. It follows that the forward problem is to construct the boundary measurements when the internal properties are known, to model propagation of current.

Electrical impedance tomography has many applications in the medical field including detection of abnormalities in lungs and monitoring breathing or blood flow but the method can also be used in industrial applications, for instance in oil and gas exploration as well as structural analysis. There are however challenges to overcome before the method is viable in practice.

One category of these problems has to do with the algorithms used in reconstructing the conductivity distributions from boundary measurements. Compared to computed tomography or magnetic resonance imaging, the main benefits of electrical impedance tomography are time and contrast resolution [20]. Many of the proposed applications rely on the ability to generate images quickly and their adoption requires that this is possible not only in theory but in practice as well.

While the forward problem is intrinsically easier to solve than the inverse problem, these same requirements apply to algorithms designed to solve the direct problem. This is most

prominent in algorithms that solve the inverse problem by first solving the forward problem.

Traditionally there have been two ways to solve the forward problem, analytically and with numerical methods. Analytical solutions have the benefit that they are efficient, but at the same time have limited practical use as solutions exist only for a small number of idealized geometries [4]. While numerical methods provide a way to represent arbitrary geometries, they are computationally more demanding.

Proposed numerical methods include linear methods that assume that conductivity of the domain is close to a constant, iterative methods that are fast but not very accurate and layer-stripping methods that address the full non-linear problem but do not work well on complete electrode model data.

As artificial neural networks are able to approximate non-linear continuous functions with arbitrary precision while at the same time being computationally efficient and allowing concurrent computation, they could prove to be a fast and accurate way to approximate the solutions to the problems associated with electrical impedance tomography. These neural solutions are not constrained to constant conductivity distributions or certain shaped meshes.

Particularly graph neural networks prove to be a viable candidate to extend the set of numerical methods.

Unlike more traditional neural network solutions graph neural networks have the benefit that they are mesh invariant and therefore the input of the model does not have to be the one used in training allowing for more general solutions. Multilayer perceptron networks are only able to learn mappings between finite-dimensional Euclidean spaces, while graph neural networks have been shown to be able to converge on mappings between infinite-dimensional spaces. This means they can learn to approximate the partial differential operators which govern among other things the electrical impedance tomography forward problem.

At the same time graph neural networks enjoy many of the computational benefits of other neural network structures as they rely on the same matrix computations that allow batch operations, distributing calculations between multiple processing units and GPU-acceleration.

In electrical impedance tomography denser meshes offer more accurate results but at the same time increase processing time. Therefore neural networks could prove beneficial in extending the set of existing methods of solving the forward and inverse problems associated with it.

It has also been shown that the architecture of the standard graph neural networks we

discuss in this thesis can be extended to achieve linear time complexity [17], making the method even more appealing. Further, neural network based methods do not require any knowledge of the underlying partial differential equation and only observational data is needed. This means results are not tied to the completeness of our understanding of the governing partial differential equation.

In this thesis, we will first briefly familiarize the reader with the fundamentals of electrical impedance tomography and artificial neural networks. We will then present the method of applying graph kernel neural networks to the electrical impedance tomography forward problem and finally present experimental results on the subject.

The aim is not to provide a comprehensive understanding of neural networks or electrical impedance tomography but rather go over core concepts related to the topic. We will for instance omit training of neural networks almost completely as we are mostly interested in their ability to approximate functions after training.

Chapter 2

Electrical Impedance Tomography

Electrical impedance tomography or EIT is an imaging technique where the electrical conductivity and permittivity inside a domain is inferred from measurements of electric current and potentials at the boundary [5]. A typical imaging system consists of multiple conducting electrodes that both apply a low amplitude current to the surface of an object and take the measurements of voltage. A tomographic differential image is then formed based on the measurements.

EIT has many benefits compared to many other imaging modalities as it is non-ionizing, inexpensive and allows for near real time imaging. Among other applications, the method can be used to detect pulmonary emboli and blood clots in the lungs. The benefits of the method are made apparent in the diagnosis of the former as it requires the inhalation of radioactive gas and injection of radio-opaque dye [21] whereas EIT requires no exposure to any radioactive material. Outside the medical field the method can be used to determine the location of mineral, oil and gas deposits [22], and to detect leaks in underground storage tanks [1] and defects in metal [8].

However the challenges related to EIT, have made adoption relatively slow. First is the non-local property of conductivity imaging [12]. Unlike, for example, in x-ray imaging where radiation passing through an object is almost completely only affected by the matter on its path, current diffuses into its surroundings. A single x-ray passing through a domain affects only a small subset of measurements whereas current passing from one localized area to another can change all the measurements. This is further complicated by the fact that, following the path of least resistivity, injected current can escape anywhere inside the object. This means

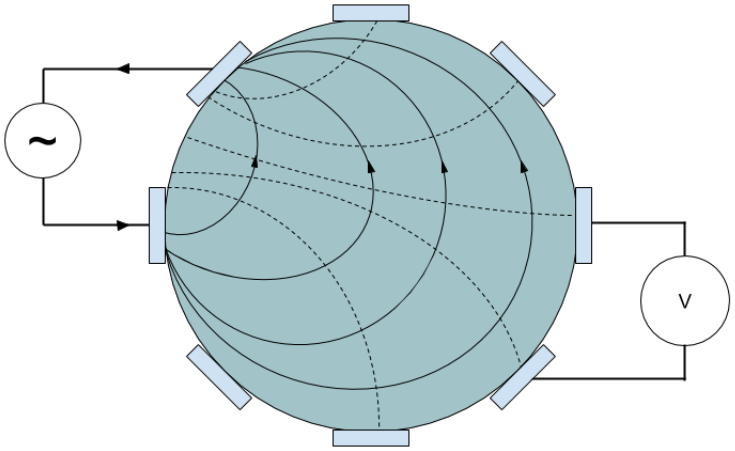


Figure 2.1: EIT setup where alternating current is injected into a circular domain on the left creating a current field and the potential field running orthogonal to it, is measured on the right as voltage.

that if we place electrodes in a circular pattern around a part of an object some of the injected current may not reach the all the electrodes and therefore the voltage can be partially or fully not measured by our measuring electrodes.

Secondly the reconstruction problem is *ill posed*, meaning in this case that given measurement data, we cannot know if the solution is unique or that it depends continuously on the data. The latter proves to be the principal issue, as there can be arbitrarily large changes in the conductivity within the domain which are undetectable by taking measurements on the boundary. We can however constrain the solutions with *a priori* information so that reconstruction is possible [12].

Both of these properties mean that constructed images are typically less clear and the

reconstruction process is more complex as well as computationally expensive than in many other tomography methods.

2.1 The EIT Forward Problem

The forward problem of EIT is to find the potential distribution on the boundary of a domain given the domain's conductivity distribution and the current injected through it's boundary. In practise, this means inferring electrode measurements on the surface of an object when we know the conductivity distribution of its interior, e.g. placement and conductivity of lungs inside a human thorax.

The problem was originally considered by A.P. Calderón in his paper "On an inverse boundary value problem" [6] where he showed that the Fréchet derivative of the mapping from the conductivity distribution to the voltage distribution on the boundary of the domain is injective when the conductivity is constant. His proof stopped short of determining that the mapping itself is injective stating that this would indeed be the case if the range of the derivative would be closed. While we are more concerned with the forward problem, Calderón was focused on the inverse problem, solving the domain's conductivity distribution when the electrical potential on the boundary is known. His goal was to determine if the voltage distribution on the boundary is injective, so that it could be proved to be invertible.

Formally we can describe the forward problem as follows. If we let Ω denote the domain, $\partial\Omega$ its boundary and f the voltage distribution on $\partial\Omega$, the mathematical model describing the relationship between voltage u and conductivity distribution σ within Ω is given by the elliptic Dirichlet boundary value problem

$$\begin{cases} \nabla \cdot \sigma \nabla u &= 0 \qquad \text{in } \Omega, \\ u &= f \qquad \text{on } \partial\Omega \end{cases} \tag{2.1}$$

when there are no current sources or sinks in Ω.

With this notation the forward problem of EIT is to recover u given σ. Conversely, the inverse problem, and the actual objective of EIT, is to recover σ when f is known.

This equation represents the idealized *continuum model* where the infinite-dimensional boundary measurements can be modelled by a *Dirichlet-to-Neumann map* to solve the problem analytically. However, practical applications rely on electrodes to gather measurements. Be-

cause of this, measurements cannot be taken over the whole boundary but instead on finitely many sections of it. The electrodes also limit our ability to inject current in a similar fashion. To address this, the boundary condition of the problem has to be refined for practical applications.

In this thesis, we will use *the complete electrode model* to achieve this.

2.1.1 Dirichlet and Neumann Problems

The types of boundary value problems that arise from EIT are Dirchlet and Neumann problems.

Dirichlet Problems

Dirichlet problems consist of finding a function that solves a partial differential equation in the interior of a domain and coincides with a given continuous function on the boundary of that domain.

When solving for u these problems are of the form

$$\begin{cases} \mathcal{L}u &= 0 \quad \text{in } \Omega, \\ u &= f \quad \text{on } \partial\Omega \end{cases} \tag{2.2}$$

where $\mathcal{L}$ is some partial differential operator and f is known.

The continuum model equation 2.1 is one example of such a problem.

Neumann Problems

Neumann problems consist of finding a function that solves a partial differential equation in the interior of a domain and has its normal derivative coincide with a given function on the boundary of that domain.

When solving for u these problems are of the form

$$\begin{cases} \mathcal{L}u &= 0 \quad \text{in } \Omega, \\ \frac{\partial u}{\partial \nu} &= g \quad \text{on } \partial\Omega \end{cases} \tag{2.3}$$

where $\mathcal{L}$ is some partial differential operator and ν is the outer normal of the boundary $\partial\Omega$.

Dirichlet-to-Neumann Maps

The Dirichlet-to-Neumann map is a function

$$\Lambda : f \mapsto \frac{\partial u}{\partial \nu}$$

defined in the EIT continuum model case 2.1 by

$$\Lambda_\sigma(f) = \sigma(\frac{\partial u}{\partial \nu})$$

where u is the solution to the Dirichlet problem. Λ_σ uniquely determines σ in Sobolev spaces with two dimensions as shown by K. Astala and L. Päivärinta in [3]. Additionally J. Sylvester and G. Uhlmann have shown in [26] that uniqueness holds in three and higher dimensional cases given that the conductivities are smooth.

2.2 The Complete Electrode Model

The complete electrode model or CEM is a way to model the elliptic boundary value problem described in equation 2.1 with only discrete and finite measurements at the boundary.

It has been stated that CEM can be seen as a finite element approximation of the continuum model [15] and has been shown to be able to approximate the continuum model up to measurement precision [7, 15, 25].

If we update our boundary conditions according to CEM to take into account the finite electrode measurements as well as their contact resistances, we can define the elliptic boundary value problem as

$$\begin{cases} \nabla \cdot \sigma \nabla u = 0 & \text{in } \Omega, \\ \frac{\partial u}{\partial \nu} = 0 & \text{on } \partial\Omega \setminus \overline{E}, \\ u + z_m \sigma \frac{\partial u}{\partial \nu} = U_m & \text{on } E_m, m \in \{1, ..., M\}, \\ \int_{E_m} \sigma \frac{\partial u}{\partial \nu} dS = I_m, & m \in \{1, ..., M\}, \end{cases} \tag{2.4}$$

where $\nu = \nu(x)$ denotes the exterior unit normal of $\partial\Omega$, $\{z_m\}_{m=1}^M$ are the contact resistances at the electrodes $\{E_m\}_{m=1}^M$ and $E = \cup_{m=1}^M E_m$. Measurements of current and potential on the boundary electrodes are denoted by $\{I_m\}_{m=1}^M$ and $\{U_m\}_{m=1}^M$ respectively. [25]

A simpler way to discretize the boundary conditions would be to only require

$$\begin{cases} \frac{I_m}{|E_m|} & \text{on } E_m \\ 0, & \text{on } \partial\Omega \setminus \cup E_m \end{cases} \tag{2.5}$$

where $|E_m|$ is the area of the electrode E_m and I_m is the current at the same electrode. The problem with this so called *gap model* is that it overestimates the resistivity of the domain because it ignores the fact that the metal electrodes themselves provide a low resistance path for the current [25].

The CEM takes this so called shunting or shorting effect into account and provides us the most accurate way of modelling the described interaction.

When current is conducted into the domain via electrodes, so called current or injection patterns are used. These patterns determine which electrodes have opposite polarities. Typically multiple different current patterns are used so that more measurements are generated allowing better spatial resolution in the reconstruction of the conductivity distribution.

If there are N electrodes, there will be $N-1$ linearly independent current patterns as one of the electrodes is considered the ground to which other potentials are compared to.

Chapter 3

Artificial Neural Networks

Artificial neural networks or *ANNs* are machine learning models that consist of processing units and connections between those units, sometimes referred as *neurons* and *synapses* respectively.

As the terminology suggests, ANNs are inspired by the neurons of the natural brain. Their development has since deviated from this starting point and a large variety of models have been proposed. Due to this variety, models can have fundamental differences outside the neuron-synapse structure and further generalizations of the definition tends to lead to some ambiguity.

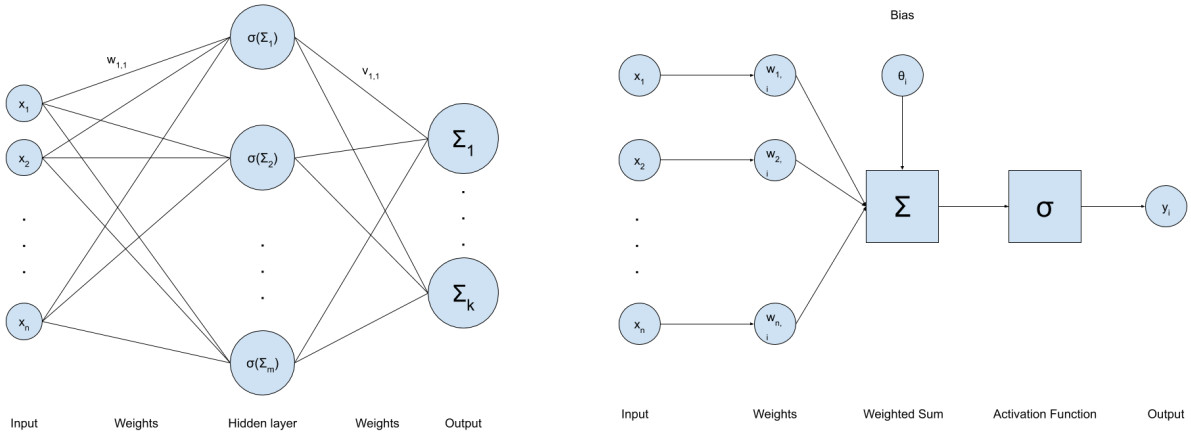
We will therefore use the *multilayer perceptron* or *MLP* as a prototypical ANN. MLPs are often somewhat broadly used synonymously with neural networks and provide a straightforward way into understanding interactions between neurons and their connections. Many of the more complex models are built on the fundamentals of MLPs.

We will first familiarize the reader with artificial neural networks using multilayer perceptrons and later expand into *graph neural networks* and *graph kernel networks*.

3.1 Multilayer Perceptrons

A multilayer perceptron consists of neurons arranged into layers. Each layer is connected to the next and information only moves in one direction, from the input towards the output. We call these networks where the connections between neurons do not form cycles *feedforward neural networks*. This is opposed to *recurrent neural networks* where the connections form a directed graph and aren't strictly restricted between adjacent layers.

Connections in MLPs can either run between all the neurons of adjacent layers or between



Multilayer perceptron Artificial neuron

Figure 3.1: Structure of a dense multilayer perceptron and a single neuron

just some of them. We will focus on the fully or *densely* connected case because it is mathematically more convenient and allows connections to disappear when appropriate by setting weights to zero.

It is worth noting that in practice this zeroing out connections does not make dense MLPs a more general case when compared to sparse MLPs which are described for instance by Mocanu et al. in [19]. The weight matrices of dense networks are still the same size and no direct benefit is gained from the point of view of computational complexity whereas sparse networks offer often improvements in regard of both space and time complexity as less connections need to be taken into account in computations.

In an MLP, a neuron takes a sequence of input values and applies a real valued function to their sum. The result of which is then outputted by the neuron. This function is called an *activation function*. Typically a non-linear activation is used so that the network is able to approximate non-linear functions.

The connection between neurons is defined by the synaptic weight. These weights determine the relative importance of an input to the neurons it is connected to. During the training process these weights are updated or "learned". Broadly, this is done by typically comparing an output of a network with a ground truth or target value, applying an error function to these two values and computing the gradient of the error function. Weights are then updated so that output is

shifted in the opposite direction of the gradient.

The first layer of the network is referred to as the input layer and the last as the output layer. The layers in between these two are called hidden layers.

When there is a single hidden layer with $N \in \mathbb{N}$ neurons and a singular output node, we can represent these networks as sums $\tau(x)$ such that

$$\tau(x) = \sum_{i=1}^{N} w_i \sigma(y_i x + \sigma_i),$$

where $w_i \in \mathbb{R}$ is the weight coefficient between the neuron $j \in \{1, ..., N\}$ and the output layer, $x \in \mathbb{R}$ is the input, y_i is the weight coefficient between the input and the neuron i, $\theta \in \mathbb{R}$ is the bias and $\sigma(t)$ is some bounded and non-constant function.

It has been shown by Hornik in [13] that if N is some finite but sufficiently large number, the set of these functions is dense in the set of continuous functions. Therefore MLPs are capable of approximating continuous functions with arbitrary precision. In Hornik's paper the network is built with only a singular output node and therefore the codomain of τ is $\mathbb{R}$. However the proof of the universal approximation theorem allows the result to be extended into $\mathbb{R}^n$, $n \in \mathbb{N}_{\geq 2}$ [14].

This *universal approximation property* guides many applications of neural networks as it allows continuous functions to be substituted with MLPs. This approach can be seen in the following descriptions of graph neural networks and graph kernel networks. It also plays a fundamental role in the reasoning why graph kernel networks are capable of approximating partial differential operators.

As the name multilayer perceptron suggests, a *perceptron* is an MLP without any hidden layers. Traditionally perceptrons have been binary classifiers with a scalar output and sometimes the term multiclass perceptron is used to differentiate between one and multiple dimensional cases. However, in this thesis we will use the term perceptron as mentioned, to describe any feedforward network where the input is connected directly to the output without requiring that it is a strictly a classifier.

3.2 Graph Neural Networks

Graph neural networks or GNNs are neural networks that can directly process most types of graphs. As a large variety of data such as natural language, chemical compounds and particle systems can be modelled with graph structures, extending the MLP model to take this into account can intuitively be seen as beneficial.

The working idea behind graph neural networks is that concepts are not defined by only the features they have but also by other concepts they are related to. In graphs we can represent concepts with nodes and their relationships with edges. This type of ANN has the benefit, that unlike many other types of neural networks, we do not have to pre-process graph structured data and map it to a simpler representation, possibly losing information in the process. For instance, the input of an MLP has to be an n-dimensional vector and by flattening the graph, we risk losing information on the topological dependency of the nodes [23].

There are two stages to mapping a graph to an output with a GNN. First there is the message passing or propagation step that computes the representation for each node describing its relationship with other nodes in the graph. Second step is to map these representations to an output.

3.2.1 The Graph Neural Network Model

In the center of GNNs is the notion of *states*. We can attach, to each node n, a state $x_n \in \mathbb{R}^d$ that contains information about the node and its neighbourhood. More concretely x_n is the vector representation of the concept we denoted by the node n.

Now, if we let f_w be a parametric *local transition function* that expresses node n's dependence on its neighbourhood and let g_w be the *local output function* that produces the output, then we can define a state x_n and an output o_n that represents the inference about the node n as

$$\begin{aligned} x_n &= f_w(l_n, l_{co[n]}, x_{ne[n]}, l_{ne[n]}) \\ o_n &= g_w(x_n, l_n) \end{aligned} \tag{3.1}$$

where l_n is the label of n, $l_{co[n]}$ are the labels of edges connected to n, $x_{ne[n]}$ are the states of the nodes in n's neighbourhood and $l_{ne[n]}$ are the labels of nodes in n's neighbourhood.

The representation 3.1 does not take into account direction of edges but the function f_w can be extended with a parameter that denotes the direction of each edge. The GNN is therefore

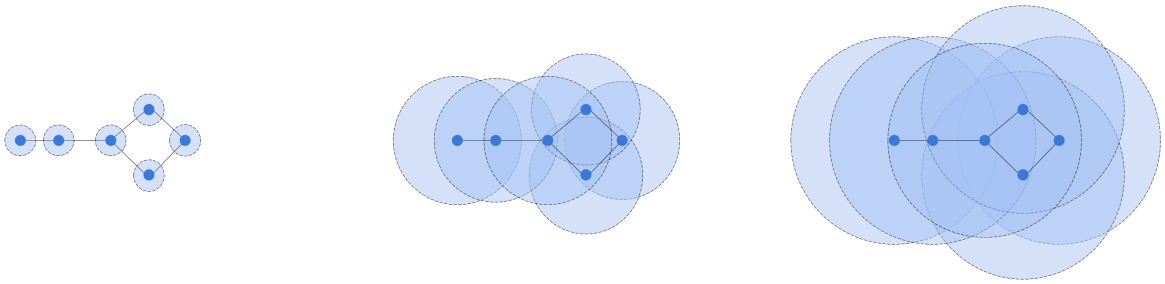
equipped to handle cyclic, acyclic, directed and undirected graphs.

After we have defined these local functions, we combine them into vectors x, o, l and l_N that contain all the states, outputs, labels and labels of each node, respectively. We can then write 3.1 in the more compact form

$$\begin{aligned} x &= F_w(x, l) \\ o &= G_w(x, l_N) \end{aligned} \tag{3.2}$$

where F_w is the *global transition function* and G_w is the *global output function*. These global functions are N-dimensional vectors of the local f_w and g_w functions.

3.2.2 Message Passing in Graph Neural Networks



t=0 t=1 t=2

Figure 3.2: Synchronous message massing between nodes at different time steps growing the perceptive fields of nodes and learning their position in the graph

GNNs are based on message passing. States are updated based on the states of their neighbouring nodes. Information on the states is exchanged until a stable equilibrium is reached. There are many implementations of different message passing networks similar to the archetypical GNN described in [23] that do not require that the time steps are advanced until convergence [10]. In these networks time steps are addressed as a hyperparameter and are increased for a finite number of times.

Banach's fixed point theorem (7.2) however states that there does exist a unique solution to the equation 3.1 if F_w is a *contraction map* (7.1) with respect to the state.

The theorem also suggests an iterative way to compute the state

$$v_{t+1}(x) = F_w(v_t(x), l) \tag{3.3}$$

where $v_t(x)$ denotes the t^{th} iteration of x. This iterative computation method converges exponentially fast to the solution of equation 3.2 [23]. As the model implements the *Jacobi iterative method*, we can compute the states using the local functions from equation 3.1 by iterating

$$\begin{aligned} v_{t+1}(x_n) &= f_w(l_n, l_{co[n]}, x_{ne[n]}(t), l_{ne[n]}) \\ o_n(t) &= g_w(v_t(x_n), l_n) \end{aligned} \tag{3.4}$$

this can be seen as a network of units that compute f_w and g_w. We will call such a network an *encoding network*.

We can construct the encoding network by replacing the nodes in the original graph with units that compute the function f_w. Each unit stores the current state of the node $v_t(x_n)$ and calculates its next state $v_{t+1}(x_n)$ when the time step t is advanced. These calculations are done simultaneously and all states transfer from state $v_t(x)$ to $v_{t+1}(x_n)$ at the same time. Units that compute g_w are constructed in the same way.

We can implement f_w and g_w as feed forward neural networks in which case the encoding network turns out to be a recurrent neural network. Connections between the neurons of the network can be divided into internal and external connections where the internal connections are determined by the neural network structure and the external connectivity depends on the edges of the original graph.

3.2.3 Training the Graph Neural Network

Because GNNs can be unravelled into recurrent neural networks and as such are completely differentiable, they can be trained using gradient based methods. Learning can be posed as the minimization problem of a quadratic loss function

$$e_w = \sum_{i=1}^{p} \sum_{j=1}^{q_i} (t_{i,j} - \tau_w(G_i, n_{i,j}))^2 \tag{3.5}$$

where $t_{i,j} \in \mathbb{R}$ is the target value, $n_{i,j}$ is a supervised node in the graph G_i, q_i is the number of supervised nodes in a graph, p is the size of the dataset and τ_w is the function representation of a GNN. The goal is to optimize the parameter w.

Compared to other neural network models the learning process differs slightly with GNNs due to the fact that we have to advance the time step before computing the gradients. We iteratively compute the states $v_t(x_n)$ until the time T when the states approach the fixed point solution $v_T(x) \approx x$.

After this process we are free to calculate the gradient $\frac{\partial}{\partial w} e_w(T)$ and update weights w accordingly.

3.3 Discussion on Network Models

So far the neural networks we have discussed are mappings to finite-dimensional Euclidean spaces i.e.

$$\tau_k(x) : D \to \mathbb{R}^m$$

where $\tau_k(x)$ is the function representation of neural network k and D is its domain. For MLPs D is $\mathbb{R}^n$ and for GNNs it is some graph $\mathcal{G}$.

This poses some limitations to their usage, as the networks can only output finite-dimensional vectors. For instance, when training a neural network to approximate a partial differential equation, we would have the network approximate the functions involved instead of their answers. This, however, would require that the codomain of the network is a function space.

It is to be noted, that we can compensate for these limitations and often still arrive at useful results. Considering the case of partial differential equations. With the presented neural networks, we can parametrize the operator as a neural network. This approach, however, is not mesh independent and we will need to modify the network if we want to change the domain or the resolution [16]. Another approach is to parametrize the solution to the partial differential equation as a neural network. This method, while mesh independent, requires a new neural network for each new coefficient function. This method is very similar to the finite element method and suffers from same computational issues. In addition, the underlying partial differential equation must be known. [16]

To overcome these limitations, we can extend the concept of GNNs further to *graph kernel networks*. These models presented by Li et al. in [16] are able to learn mapping between infinite-dimensional spaces. They are mesh invariant and solve many computational issues of the presented methods.

3.4 Graph Kernel Networks

Unlike the previously presented methods, graph kernel networks or graph kernel networks are able to learn mappings between function spaces instead of just Euclidean spaces. This means the networks are not restricted by the dimensionality of their codomain.

Following the approach taken by Li et al., when applied to partial differential equations graph kernel networks are able to transfer solutions between meshes and do not necessarily require additional training when applied to different problems. They also do not require any knowledge of the underlying partial differential equations and can therefore be trained on only experimental or simulation data. [16]

3.4.1 The Graph Kernel Network Model

Graph kernel networks follow the architecture of GNNs closely with the exception on how the states of the nodes are computed. The contribution of graph kernel networks is to calculate the states using an integral kernel or a *neural operator*.

The same iterative approach is used when computing the states but we define

$$v_{t+1}(x) = F\left(W_{v_t(x)} + \int_D \kappa_\theta(\cdot) v_t(y) \nu_x(dy)\right), \tag{3.6}$$

where $F : \mathbb{R} \to \mathbb{R}$ is a fixed function applied elementwise, ν_x is a fixed Borel measure for each $x \in D$, $W \in \mathbb{R}^{n\times n}$ is a matrix, θ are parameters for the kernel $\kappa_\theta : \mathbb{R}^{2(d+1)} \to \mathbb{R}^{n\times n}$. W, θ and κ_θ are learned from data.

If we replace the Borel measure ν_x by an empirical approximation based on K grid points, we may view κ_θ as a $K \times K$ kernel block matrix where each entry $\kappa_\theta(x, y)$ is a $n \times n$ matrix that shares the same set of network parameters. This allows developing a method that shares parameters independent of the discretization used [16]. In practise the kernel κ_θ is modelled with a neural network $\tau : \mathbb{R}^{2(d+1)} \to \mathbb{R}^{n\times n}$.

In practice, the message passing step 3.6 can be defined as

$$v_{t+1}(x) = F\left(W_{v_t(x)} + \frac{1}{N(x)} \sum_{y\in N(x)} \kappa(e(x, y)) v_t\right), \tag{3.7}$$

where $W \in \mathbb{R}^{n\times n}$, $N(x)$ is the neighbourhood of x,$\kappa_\phi(e(x, y))$ is a neural network taking edge

features as input and outputting a matrix in $\mathbb{R}^{n\times n}$. This form can be thought of a Monte Carlo approximation of 3.6.

3.4.2 Approximating Partial Differential Equations with Graph Kernel Networks

To achieve the goal of learning mappings between two infinite dimensional spaces, Li et al. used partial differential equations as a guiding principle in building the graph kernel network architecture. Applying graph kernel networks as approximations of partial differential operators is therefore not only possible but also seminal.

The graph kernel network architecture is based on the following example. Let $\mathcal{L}_a$ be a partial differential operator depending on a parameter a such that $\mathcal{L}_a u = -\nabla \cdot (a\nabla u)$ and define a partial differential equation

$$\begin{cases} \mathcal{L}_a u &= f \qquad \text{in } \Omega, \\ u &= 0 \qquad \text{in } \partial\Omega \end{cases} \tag{3.8}$$

for a bounded, open set $\Omega \in \mathbb{R}$ and some fixed function f. Now may define the Green's function $G : D \times D \to \mathbb{R}$ as the unique solution to the problem

$$\mathcal{L}_a G(x, \cdot) = \delta_x$$

where δ_x is the *Dirac measure* on $\mathbb{R}^d$ centred at x. As G depends on a we will denote it as G_a. Now

$$\begin{aligned} \mathcal{L}_a u(x) &= \mathcal{L}_a \int_\Omega G_a(x, \cdot)(y) f(y) dy \\ &= \int_\Omega \mathcal{L}_a G_a(x, \cdot)(y) f(y) dy \\ &= \int_\Omega \delta_x(y) f(y) dy \\ &= f(x) \end{aligned} \tag{3.9}$$

From this follows a representation of the solution u as

$$u(x) = \int_\Omega G_a(x, y) f(y) dy.$$

Now, Green's function $G(x, y)$ is generally continuous at points $x \neq y$ when $\mathcal{L}_a$ is uniformly elliptic and is therefore well suited to be modelled with a neural network due to the universal approximation property. We then arrive at the model described at equation 3.6. In the equation the approximation of the Green's function is done with the network κ.

When compared to other networks graph kernel network require far fewer samples to approximate at least some partial differential equations. For example a graph kernel network was able to achieve smaller test error with just 5 training samples than a dense neural network with 100 samples when approximating the 2-dimensional Poisson equation [16]. This is because they are capable of learning the Green's function associated with the solution of the partial differential equation.

Chapter 4

Applicability of Graph Kernel Networks to the EIT Forward Problem

Using graph kernel networks to solve the EIT forward problem depends on their ability to approximate solutions to partial differential equations with non-zero Dirichlet boundary conditions. As stated in [16], this however does not appear to pose a problem.

We can apply the same reasoning as in equation 3.9 to problems with non-zero Dirichlet boundary conditions with the representation formula using Green's function which states that, if $u \in C^2(\bar{U})$ solves

$$\begin{cases} -\Delta u &= f \text{ in } \Omega \\ u &= g \text{ on } \partial\Omega, \end{cases} \tag{4.1}$$

then

$$u(x) = \int_\Omega G(x,y)f(y)dy - \int_{\partial\Omega} \frac{\partial G(x,y)}{\partial \nu} g(y) dS(y) \tag{4.2}$$

This form allow us to observe that if u is harmonic in Ω, i.e $\Delta u = 0$ in Ω, then

$$u(x) = -\int_{\partial\Omega} g(y) \frac{\partial G(x,y)}{\partial \nu} dS(y)$$

Which allows us to utilize the neural network integral kernel to approximate the solution similarly to 3.6 as the derivative of Green's function is generally continuous when $x \neq y$ and the boundary is not too pathological.

It should be noted that the above only applies to the continuum model as the complete electrode model is not given in the form of a Dirichlet problem. Therefore the reasoning does

not directly apply to the latter model and the following proof does not prove applicability in a practical setting. The analysis does however offer insight into how graph kernel networks work with regard to Green's functions.

While this holds, graph kernel network solutions can also be used readily in complete electrode model case, as implied by our experimental results with simulated data.

Proof. Following largely the proof of Evans in [9], suppose that $u \in C^2(\bar{U})$ is an arbitrary function. Fix $x \in \Omega$ and $\epsilon > 0$ such that $B(x, \epsilon) \subset \Omega$, and define $V_\epsilon := \Omega \setminus B(x, \epsilon)$. Therefore

$$\begin{aligned} &\int_{V_\epsilon} u(y)\Delta\Phi(y-x) - \Phi(y-x)\Delta u(y)dy \\ &= \int_{\partial V_\epsilon} u(y)\frac{\partial\Phi(y-x)}{\partial\nu} - \Phi(y-x)\frac{\partial u(y)}{\partial\nu}dS(y), \end{aligned} \tag{4.3}$$

where ν denotes the outer unit normal vector on ∂V_ϵ and $\Phi(\cdot)$ is a the fundamental solution of Laplace's equation i.e.

$$\Phi(x) = \begin{cases} -\frac{ln|x|}{2\pi} & n = 2 \\ \frac{1}{n(n-2)\alpha(n)}\frac{1}{|x|^{n-2}} & n \geq 3. \end{cases}$$

Now, as $\Delta\Phi(x-y) = 0$ for $x \neq y$, we see that

$$|\int_{\partial B(x,\epsilon)} \Phi(y-x)\frac{\partial u(y)}{\partial\nu}dS(y)| \leq C\epsilon^{n-1}\max_{\partial B(0,\epsilon)}|\Phi| = o(1), \text{ as } \epsilon \to 0.$$

Further,

$$\int_{\partial B(x,\epsilon)} u(y)\frac{\partial\Phi(y-x)}{\partial\nu}dS(y) = \fint_{\partial B(x,\epsilon)} u(y)dS(y) \to u(x), \text{ as } \epsilon \to 0.$$

Therefore, if we let $\epsilon \to 0$ in 4.3, we get

$$u(x) = \int_{\partial\Omega} \Phi(y-x)\frac{\partial u(y)}{\partial\nu} - u(y)\frac{\Phi(y-x)}{\Phi\nu}dS(y) - \int_\Omega \Phi(y-x)\Delta u(y)dy \tag{4.4}$$

Now we can introduce for the fixed x a function $\phi^x = \phi^z(y)$, solving the boundary value problem

$$\begin{cases} \Delta\phi^x = 0 & \text{in } \Omega \\ \phi^x = \Phi(y-x) & \text{on } \partial\Omega \end{cases} \tag{4.5}$$

Applying Green's formula once more, we have

$$-\int_\Omega \phi^x(y)\Delta u(y)dy = \int_{\partial\Omega} u(y)\frac{\partial\phi^x(y)}{\partial\nu} - \phi^x(y)\frac{\partial u(y)}{\partial\nu}dS(y) = \int_{\partial\Omega} u(y)\frac{\partial\phi^x(y)}{\partial\nu} - \Phi(y-x)\frac{\partial u(y)}{\partial\nu}dS(y)$$

Combining this with 4.4, we get

$$u(x) = -\int_{\partial\Omega} u(y)\frac{\partial G(x,y)}{\partial \nu}dS(y) - \int_{\Omega} G(x,y)\Delta u(y)dy, \tag{4.6}$$

where $G(x,y) := (\Phi(y-x) - \phi^x(y)$ is the Green's function for the region Ω and

$$\frac{\partial G(x,y)}{\partial \nu} = D_y G(x,y) \cdot \nu(y),$$

is the outer normal derivative of G with respect to y.

Now if we assume that $u \in \Omega$ solves 4.1 and insert it into this equation, we arrive at

$$u(x) = \int_{\Omega} G(x,y)f(y)dy - \int_{\partial\Omega} \frac{\partial G(x,y)}{\partial \nu} g(y)dS(y) \tag{4.7}$$

□

Chapter 5

Experiments

In this chapter we outline our experimental setting and illustrate the capabilities of graph kernel networks in solving the EIT forward problem.

As a note, we will be discussing multiple types of nodes in this chapter as both the finite element meshes used to simulate training data and the input graph of the neural network contain nodes. We will try to make clear from context which class of nodes we are talking about and will mostly refer to the finite element mesh nodes specifically as such.

5.1 Experimental Setting

Experimentation was conducted by simulating conductivity distributions and solving them using the finite element method.

Graphs were then constructed from conductivity distribution meshes and a graph kernel network was trained on them using the voltage distributions within the domain as ground truth. These target values were acquired by computing them using the FEM.

This approach allows us to synthesize training data with little effort and gives us relatively easy access to the conductivity distributions that generate each of the boundary measurements. However, the method has the downside that the global minimum for the network's loss function is not achieved by the model converging to the actual solution to the forward problem but rather the FEM solution used to generate the target measurements. Given the degree of how much synthesizing the data simplifies the experimental setting and the fact that a network with good enough generalizing properties should approach the actual solution to the problem, the

described method of data generation was chosen.

5.1.1 Data Generation

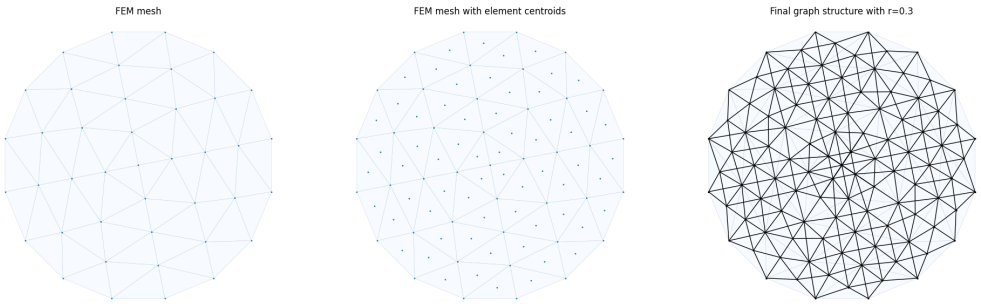


Figure 5.1: Process of creating graphs from FEM meshes. Centroids are added to encode conductivity information of each element into the network

The training data was generated with EIDORS[2], an EIT and diffuse optical tomography reconstruction software. EIDORS was used for both creating the conductivity meshes and solving the forward problem using the FEM.

The training dataset is comprised of a single homogeneous mesh. The circular mesh has radius one and 576 elements, each with conductivity of one. The mesh was modelled using 16 CEM electrodes with contact impedances of 10^{-3} spread out evenly along the circumference of the unit circle mesh. Combined electrodes cover $\frac{1}{3}$ of the boundary and the size of one electrode is therefore $\frac{1}{48}$ of the boundary. Each electrode spans two boundary nodes. Taking measurements and injecting current happens therefore over an edge rather than through a node.

For training, a single FEM mesh was used to generate multiple graphs based on injection and measurement patterns. Each combination of patterns was modelled with their own graph

resulting in a final training dataset of 200 graphs.

Current propagation within the mesh was simulated using the FEM with an adjacent injection pattern where two adjacent electrodes form the circuit that is used to introduce current into the domain. The simulations were computed with one ampere injection current.

The input graphs were constructed on top of the FEM meshes, so to say. A node was placed to correspond each of the nodes of the FEM mesh with additional nodes placed on the centroid of the triangles of the mesh.

As an input to the neural network each of the nodes is represented by a feature vector. These vectors are constructed as follows:

$[x_1, x_2, x_3, x_4]$, where

$$x_1 = \begin{cases} 1, \text{ if node is not a centroid node} \\ 0, \text{ otherwise} \end{cases}$$

$$x_2 = \begin{cases} 1, \text{ if node is not a centroid node} \\ c_n, \text{ otherwise} \end{cases}$$

$$x_3 = \begin{cases} 1, \text{ if node is the positive terminal in current pattern} \\ 0, \text{ otherwise} \end{cases}$$

$$x_4 = \begin{cases} 1, \text{ if node is the negative terminal in current pattern} \\ 0, \text{ otherwise.} \end{cases}$$

Here c_n is used to denote the conductivity of the triangle in which the centroid node n is placed. In addition to these node-specific feature vectors the network receives as input an adjacency matrix representing connections between nodes. As with the FEM meshes used to construct the input graphs, electrodes span two boundary nodes. Therefore nodes with the electrode designations, x_3 and x_4, always appear in pairs of adjacent boundary nodes. The euclidean distance of between connected nodes is given as an edge feature.

Only the non-centroid nodes are supervised. This is done mainly to facilitate interoperability between the graph kernel network and EIDORS, as the latter by default maps voltage distributions to nodes of the FEM mesh instead of its elements and these FEM mesh nodes correspond to the non-centroid nodes of our graph.

To learn the partial differential operator in 3.6, we defined $\nu_x(dy) = \mathbb{1}_{B(x,r)}$ as per Li et al. This allows for both efficient computation and exploits the decay property of Green's function [16]. In practise this means that connections between nodes are determined by their ball

neighbourhood with given r. In our experiments we used the value $r = 0.1$.

5.1.2 EITNet

The graph kernel network used in the experiments was constructed using Tensorflow and Spektral [11] as a graph convolution framework. For convenience, and partly convention, we will refer to the constructed network as EITNet. EITNet takes as an input a graph where each node is represented by a four-dimensional feature vector and outputs a graph where nodes are represented by a scalar. Input and output graphs have equal number of nodes.

EITNet consists of an edge-conditioned graph convolutional layer as described by Simonovsky and Komodakis in [24] and two encoding perceptrons. The purpose of these encoding networks is to first expand the feature vectors to larger dimensional embeddings to allow for richer convolution layer and finally to limit the dimension of the embeddings to one so that values can be interpreted as the voltage at each node. The convolution layer utilizes a leaky linear rectifier activation function [18] where as the encoding networks have linear activation. Time step of the convolution layer is advanced six time.

In EITNet, the kernel described in 3.7 is represented by a neural network with two hidden layers of 512 neurons each.

This fully convolutional architecture retains the benefits of GNNs, including mesh invariance, allowing the network to learn infinite dimensional mappings as discussed in chapter 3.

During our experimentation we also tested a black box variant of the network. By attaching an MLP to the topmost layer, we can allow the network to only create internal representations of the distribution it is trying to model and finally output a single value that is an approximation of the measurement taken at the measurement electrodes based on a given conductivity distribution and injection current.

While we found this black box method to have good convergence properties, it was relatively slow as the output MLP took all the feature vectors of the last graph layer as an input and restricted in the amount of nodes in the graph it could handle as the input of an MLP has to be of fixed in size. Both of these problems could be solved with pooling methods at the last graph layer but in our experiments this resulted in poor performance in our experiments, likely due to losing spatial information.

The final architecture of EITNet allows us to extend the model into a black box variant if wished. As the injected current and interior voltage distribution can be used to solve boundary

measurements, it should be possible to utilize transfer learning and combine EITNet with an MLP to output measurements instead of voltage distributions.

5.2 Experimental results

In our experiments, we found that EITNet is able to reproduce interior voltage distributions with an extremely small dataset and limited computational resources giving promise to further research.

As we can see from figure 5.2, our network is able to achieve relatively close approximation of the target solution even with very light optimization of hyperparameters and limited computational resources.

While the current EITNet solutions exhibit clear artefacting, this does not seem to critically affect solutions and EITNet can be said to have converged on the partial differential operator in at least some extent. We can see this from the figure 5.3.

Pictured here on the left is a conductivity distribution generated with EIDORS together with a differential image based on EITNet predictions. The image on the right is generated by subtracting an EITNet prediction made from a homogeneous mesh from one made from the conductivity distribution shown on the left. From this we can see that the propagation of current within the domain is at least somewhat accurate as the approximate location of the simulated inclusion matches the one of the input conductivity distribution.

Interestingly, it should be emphasized that the model has not been trained on samples that have inclusions within the domain, only on a single homogeneous base mesh. This suggests that the GKN network structure is very resilient against overfitting.

These results do not mean that the network described in this thesis has completely converged on the partial differential operator, as it should in the discussed ideal case. While the results are promising and EITNet can be used with different density meshes as well as different injection patterns, predictions in those circumstances are still very inaccurate and do not produce differential images anywhere near as clear as the ones shown here. All presented solutions were made with the same injection pattern and mesh density that was used in training the model.

Code used to generate EITNet, pre-trained weights and training data are made available at `https://github.com/alekmus/Graph-kernel-network`

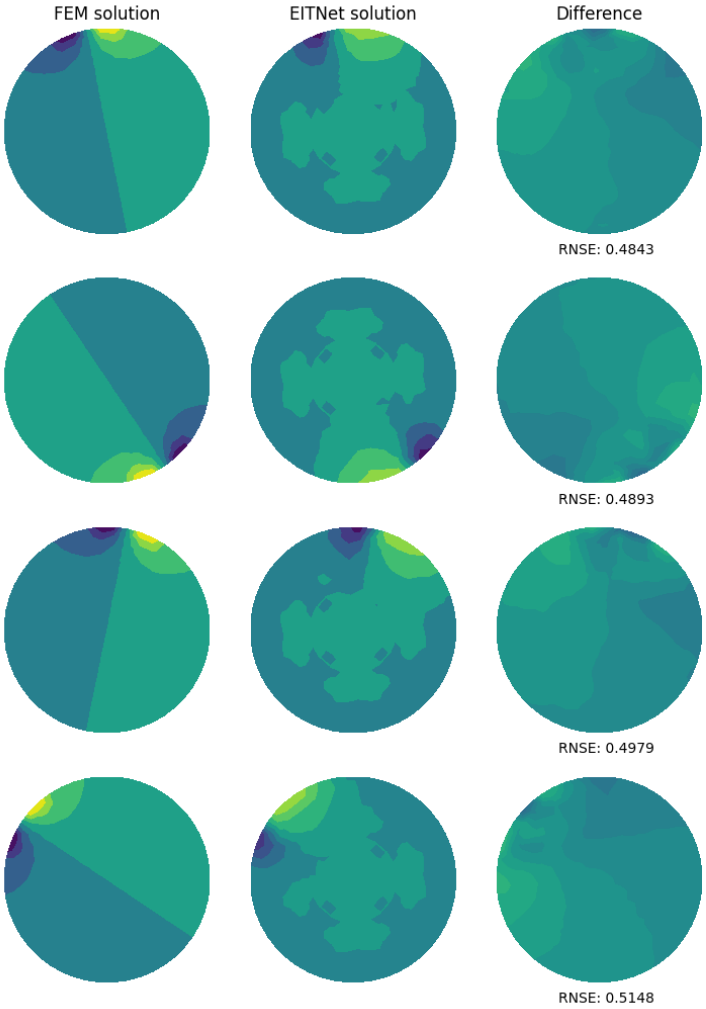


Figure 5.2: Samples from validation data comparing EITNet and finite element method solutions. Images are on the same color scale and results are calculated on a homogeneous mesh with conductivity one. RNSE denotes the relative square-norm error between the EITNet and FEM solutions.

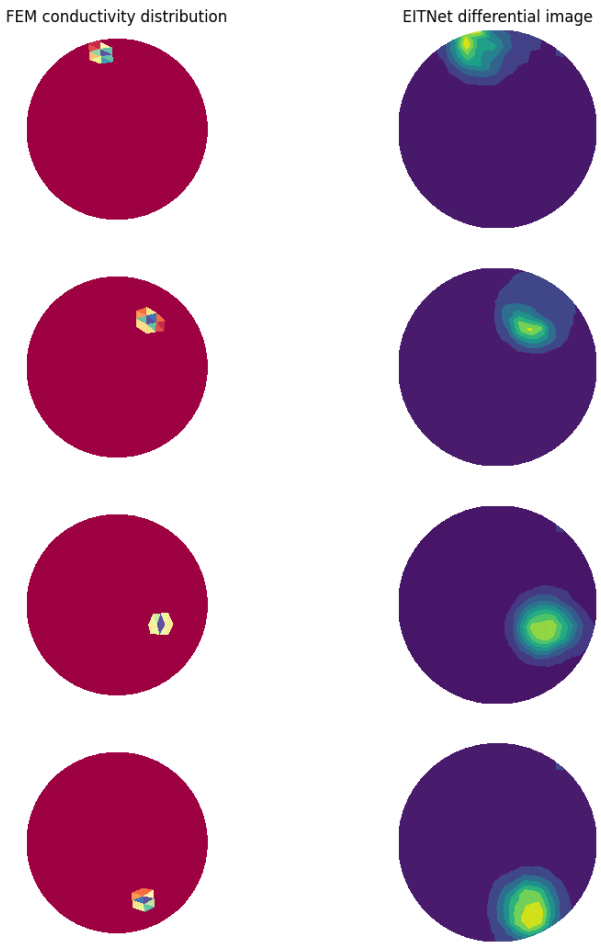


Figure 5.3: Differential image of predictions and homogeneous mesh on the right and the conductivity distribution on the left.

Chapter 6

Conclusions

In this chapter, we discuss results from the experiments and benefits as well as disadvantages of using graph kernel networks to solve the EIT forward problem.

6.1 On Benefits and Disadvantages of the Graph Kernel Network Solution

One of primary benefits of graph neural networks in the context of partial differential equations, the ability to learn the governing PDE of a phenomenon without any underlying knowledge of it, is somewhat moot when it comes to the EIT forward problem. We already have relatively sound understanding of the equation as our ability to solve the problem using the finite element method shows.

Taking this into account, neural networks can still provide a practical alternative to the more traditional methods of solving the problem. Many finite element method algorithms scale poorly with PDE system complexity and mesh density, whereas neural networks have been shown to be relatively efficient as the related matrix computations can utilize batch computation and external hardware accelerators. Additionally, as graph neural networks are not bound by the same limitations as standard MLPs, as discussed in chapter three, they are mesh invariant and the same network can be used to solve the same problem in different density meshes. This also allows us to reduce training effort by training the network on a mesh with less nodes while still benefiting from the additional information of a denser mesh during inference.

This, however, brings us to the main disadvantage of neural network solutions, the training. Unlike with finite element methods, neural networks require large amounts of data in order to converge to any meaningful solution.

It remains to be seen how much data a practical version of EITNet will need but GKNs do seem to have fairly beneficial properties in this regard as even with one example overfitting seems minimal. The presented EITNet version took only 2 hours 20 minutes to train and a single example mesh. This seems promising as collection of conductivity and voltage distribution data will likely be the largest hurdle in application of neural network methods in EIT. If we can create working examples with only a few example distributions, practical implementations could be feasible.

6.2 Future work

A natural path for further research into using graph kernel networks in EIT would be to apply the ideas Li et al. have put forward in their more general research concerning the use of graph kernel networks in solving partial differential equations [16, 17]. These include among other things Nyström approximation of the integration kernel and using multipole variants of graph kernel networks.

While graph kernel networks are relatively computationally efficient, the number of edges in graphs scale with complexity $O(n^2)$. Nyström approximation of the kernel overcomes this issue. When using the method, a random sub-graph is created by uniformly sampling from the initial graph. This process is repeated number of times each time with the same sample size. These sub-graphs are used while training leading to scaling of $O(lm^2)$ where m is the number of nodes in a sub-graph and l is the number of sub-graphs. This reduced complexity comes with only a slight disadvantage as Nyström approximation leads to error scaling at $O(\frac{1}{\sqrt{m}})$ as shown in [16].

One of the main challenges for graph kernel networks while approximating partial differential equations is the fact the they usually ignore long-range interactions due to poor scaling in computational complexity when the number of nodes is increased. Li et al. strive to solve this problem in [17] where they present a new neural network model inspired by previous fast multipole methods. This multipole network introduces so called *inducing nodes* which form their own sub-graph within the initial graph to facilitate long-range communication between

nodes without directly adding edges between existing nodes. This, in essence, introduces a multiresolution graph where long range dependencies can be modelled with less computational cost.

Outside updating the algorithm, applying graph kernel networks to the EIT inverse problem could prove beneficial. In addition to the computational benefits of neural networks and their promising performance solving partial differential equations, the inherent probabilistic nature of neural networks warrants further study in this application. As mentioned the inverse problem is severely ill-posed as solutions are not unique and do not continuously depend on data [12]. This implies that there is uncertainty when reconstructing voltage distributions from boundary measurements as each measurement can be a result of multitude of distributions. Being probabilistic models, neural networks are capable of encoding uncertainty, which in turn could lead to results worth exploring.

Chapter 7

Preliminaries

7.1 Banach's fixed point theorem

Definition 7.1 (Contraction map)**.** A function $f : M \to M$ is a contraction map if for any x and y there exists a $\mu \in [0, 1)$ such that

$$d(f(x), f(y)) \leq \mu d(x, y)$$

where $d(\cdot)$ is a metric on M.

Theorem 7.2. *Let (M, d) be a non-empty complete metric space and let $T : M \to M$ be a contraction mapping. Then T has an unique fixed point x i.e*

$$T(x) = x$$

Further, given an arbitrary element $x_0 \in M$, we can define a sequence $x_n = T(x_{n-1})$ for $n \geq 1$ from which we can recover x

$$\lim_{n\to\infty} x_n = x$$

Proof. Take an arbitrary $x_0 \in M$ and define a sequence $(x_n) = T(x_{n-1})$ where T is a contraction map. Then for all $n \in \mathbb{N}$

$$d(x_{n+1}, x_n) \leq \mu^n d(x_1, x_0).$$

Now let $m, n \in \mathbb{N}$ such that $n < m$ and note that

$$\begin{aligned} d(x_m, x_n) &\leq d(x_m, x_{m-1}) + d(x_{m-1}, x_{m-2}) + ... + d(x_{n+1}, x_n) \\ &\leq \mu^m - 1 d(x_1, x_0) + \mu^m - 2 d(x_1, x_0) + ... + \mu^n d(x_1, x_0) \\ &= \mu^n d(x_1, x_0) \sum_{i=0}^{m-n-1} \mu^i \\ &\leq \mu^n (x_1, x_0) \sum_{i=0}^{\infty} \mu^i \\ &= \mu^n d(x_1, x_0) \left(\frac{1}{1-\mu} \right). \end{aligned}$$

So for arbitrary $\epsilon > 0$

$$\mu^N < \frac{\epsilon(1-\mu)}{d(x_1, x_0)}$$

as $0 \leq \mu < 1$ for some $N \in \mathbb{N}$.

further, for $n, m > N$

$$d(x_m, x_n) \leq \mu^n d(x_1, x_0) \left(\frac{1}{1-\mu} \right) < \left(\frac{\epsilon(1-\mu)}{d(x_1, x_0)} \right) d(x_1, x_0) \left(\frac{1}{1-\mu} \right) = \epsilon.$$

Therefore (x_n) is a Cauchy sequence and by completeness of (M, d) the sequence has a limit $x \in M$. We can also note that since T is Lipschitz-continuous x must be a fixed point of T because

$$x = \lim_{n \to \infty} x_n = \lim_{n \to \infty} T(x_{n-1}) = T\left(\lim_{n \to \infty} x_{n-1} \right) = T(x)$$

Lastly, T the fixed point has to be unique in (M, d), because for any pair of distinct fixed points p_1 and p_2

$$d(T(p_1), T(p_2)) = d(p_1, p_2) > \mu d(p_1, p_2).$$

As this contradicts with the fact that T is a contraction map, the fixed point has to be unique.

□

7.2 Dirac Measure

Definition 7.3. (Dirac measure) A Dirac measure is a measure δ_x on a set X such that for a given $x \in X$ and any measurable set $A \subseteq X$

$$\delta_x(A) = 1_A(x) = \begin{cases} 0, & x \notin A \\ 1, & x \in A \end{cases}$$

Theorem 7.4. *Let σ_x be a Dirac measure on a set X and $f : X \to \mathbb{R}$ be a measurable function. Then*

$$\int_X f(y)\delta_x(y)dy = f(x)$$

Proof. Let σ_x be a Dirac measure on a set X and $f : X \to \mathbb{R}$ be a measurable function. Also define a constant function $g(x') = f(x)$ for all $x' \in X$. Note that

$$\{x' \in X : f(x) = g(x') \neq f(x')\}$$

does not contain x so g is almost everywhere equal to f. Therefore the σ_x measure of g is 0 and

$$\begin{aligned} \int f d\sigma_x &= \int g d\sigma_x \\ &= \sigma_x(X) f(x) \\ &= f(x) \end{aligned}$$

□

Bibliography

[1] Abelardo A. Ramirez and W. Daily. "Detection of leaks in underground storage tanks using electrical resistance methods: 1996 results". In: (Oct. 1996).

[2] A. Adler and W. R. Lionheart. "Uses and abuses of EIDORS: an extensible software base for EIT". In: *Physiol Meas* 27.5 (May 2006), pp. 25–42.

[3] K. Astala and L. Päivärinta. "Calderon's inverse conductivity problem in plane". In: *Annals of mathematics, ISSN 0003-486X, Vol. 163, N^{o} 1, 2006, pags. 265-299* 163 (Jan. 2006). DOI: 10.4007/annals.2006.163.265.

[4] R. Bayford et al. "Solving the forward problem in electrical impedance tomography for the human head using IDEAS (integrated design engineering analysis software), a finite element modelling tool". In: *Physiological measurement* 22 (2001).

[5] L. Borcea. "Electrical impedance tomography". In: *Inverse Problems* 18 (2002).

[6] A.P. Calderón. "On an inverse boundary value problem". In: *Seminar on Numerical Analysis and its Applications to Continuum Physics (Rio de Janeiro, 1980)*. Rio de Janeiro: Soc. Brasil. Mat., 1980, pp. 65–73.

[7] K. Cheng et al. "Electrodemodels for electric current computed tomography". In: *IEEE Transactions on Biomedical Engineering* 36 (1989).

[8] M. Eggleston et al. "The Application of Electric Current Computed Tomography to Defect Imaging in Metals". In: *Review of Progress in Quantitative Nondestructive Evaluation*. Ed. by D. Thompson and D. Chimenti. Boston, MA: Springer US, 1990, pp. 455–462.

[9] L. Evans. *Partial Differential Equations*. American Mathematical Society, 1998.

[10] J. Gilmer et al. "Neural Message Passing for Quantum Chemistry". In: (2017).

[11] D. Grattarola and C. Alippi. *Graph Neural Networks in TensorFlow and Keras with Spektral*. 2020. arXiv: 2006.12138 [cs.LG].

[12] D. S. Holder. *Electrical Impedance Tomography - Methods, History and Applications*. IOP Publishing, 2005.

[13] K. Hornik. "Approximation Capabilities of Multilayer Feedforward Networks". In: *Neural Networks* 4 (1990).

[14] K. Hornik, M. Stinchcombe, and H. White. "Multilayer Feedforward Networks are Universal Approximators". In: *Neural Networks* 2 (1989).

[15] N. Hyvönen. "Complete Electrode Model of Electrical Impedance Tomography: Approximation Properties and Characterization of Inclusions". In: *SIAM Journal on Applied Mathematics* 64 (2004).

[16] Z. Li et al. *Neural Operator: Graph Kernel Network for Partial Differential Equations*. 2020. arXiv: 2003.03485 [cs.LG].

[17] Zongyi Li et al. *Multipole Graph Neural Operator for Parametric Partial Differential Equations*. 2020. arXiv: 2006.09535 [cs.LG].

[18] A. L. Maas, A. Y. Hannun, and A. Y. Ng. "Rectifier nonlinearities improve neural network acoustic models". In: *Proc. icml*. Vol. 30. 1. Citeseer. 2013, p. 3.

[19] Decebal Constantin Mocanu et al. "Evolutionary Training of Sparse Artificial Neural Networks: A Network Science Perspective". In: *CoRR* abs/1707.04780 (2017). arXiv: 1707.04780. URL: http://arxiv.org/abs/1707.04780.

[20] J. Mueller and S. Siltanen, eds. *Linear and Nonlinear Inverse Problems with Practical Applications*. Vol. 10. Society for Industrial and Applied Mathematics, 2012.

[21] J. Newell, M. Cheney, and D. Isaacson. "Electrical Impedance Tomography". In: 41.1 (1999), pp. 85–101.

[22] R. Parker. "The inverse problem of resistivity sounding". In: *Geophysics* 49.12 (Dec. 1984), pp. 2143–2158.

[23] F. Scarselli et al. "The Graph Neural Network Model". In: *IEEE Transactions on Neural Networks* 20 (2009).

[24] M. Simonovsky and N. Komodakis. *Dynamic Edge-Conditioned Filters in Convolutional Neural Networks on Graphs*. 2017. arXiv: 1704.02901 [cs.CV].

[25] E. Somersalo, M. Cheney, and D. Isaacson. "Existence and Uniqueness for Electrode Models for Electric Current Computed Tomography". In: *SIAM Journal on Applied Mathematics* 52.4 (1992).

[26] J. Sylvester and G. Uhlmann. "A Global Uniqueness Theorem for an Inverse Boundary Value Problem". In: *Annals of Mathematics* 125.1 (1987), pp. 153–169. ISSN: 0003486X.